Rewriting Formalisms With Binding Support
Comparing Combinatory Reduction Systems and Nominal Rewrite Systems with Atom Substitution

Dominguez Alvarez, Jesus

Awarding institution:
King's College London

Rewriting Formalisms With Binding Support:

Comparing Combinatory Reduction Systems and Nominal Rewrite Systems with Atom Substitution

Jesús Domínguez Álvarez

Department of Informatics
King's College London

This dissertation is submitted for the degree of
Doctor of Philosophy

King's College London June 2017

Acknowledgements

First and foremost, I am most indebted to my supervisor, Maribel Fernández, who guided me through these four years with patience and reassurance when I most needed it. Thank you for sharing your knowledge and giving me the freedom to find my own path, as intimidating as this sometimes was. I am also grateful to my second supervisor, Christian Urban, for taking the time to explain his findings and the helpful advice on how to progress with my work. Also, I am indebted to my colleague Elliot Fairweather and the interesting discussions we had about nominal theories in particular and surviving as a Phd student in general. A special thanks to Maria Lema and Andrew Miller for all the philosophical discussions at lunch time and for preserving my sanity with their kind words and advice. Finally, I would like to thank Jamie Gabbay for the use of his LaTeX command for the *new* symbol and also James Cheney and Jordi Levy for agreeing to assess my thesis, their many useful suggestions and for pointing out Goldfarb's result.

Abstract

In this thesis, two extensions of term rewriting systems with binding support are compared: Nominal Rewrite Systems defined by Fernández, Gabbay and Mackie (Fernández and Gabbay, 2007; Fernández et al., 2004) and Combinatory Reduction Systems, a well-established higher-order rewriting formalism introduced by Klop (Klop, 1980; Klop et al., 1993).

Higher-order term rewriting is a collective name for a large number of different formalisms combining term rewriting systems and the λ-calculus, a formal system for expressing functional computation which provides to term rewriting systems a notion of binding and substitution as primitive. Nominal rewriting allows a first-order approach to binding where variables, both free and abstracted, can be directly manipulated and α-equality is formally axiomatised in the logic by means of a swapping operation and a freshness relation. Despite the differences in their underlying semantic foundations, nominal and higher-order rewriting formalisms are closely related to each other. The relationship between Combinatory Reduction Systems and other higher-order rewriting formalisms has already been well documented (see for instance (Bertolissi et al., 2006; Kop, 2012; van Oostrom and van Raamsdonk, 1994; van Raamsdonk, 1999)). In (Fernández et al., 2004), Fernández, Gabbay and Mackie presented an encoding of Combinatory Reduction Systems into Nominal Rewrite Systems showing the rewrite relation to be preserved when adding to the encoded system a set of rules simulating higher-order substitution from the λ-calculus. This immediately poses two areas of study that are addressed in this thesis, the simulation of nominal rewriting in Combinatory Reduction Systems and an extension of Nominal Rewrite Systems where higher-order substitution for nominal terms is provided as primitive. As a result, the naturality of the encoded Combinatory Reduction System in nominal rewriting is improved in the sense that the rewrite relation is not only preserved but also reflected.

This thesis aims to unify both rewriting communities by providing a concrete correspondence between Combinatory Reduction Systems and the nominal rewriting framework. There is a theoretical interest in showing such correspondence because the expressive power of nominal and higher-order rewriting is put in evidence.

Techniques to prove confluence and termination of higher-order rewriting systems were studied in (Hamana, 2010; Klop et al., 1993; Mayr and Nipkow, 1998) amongst others.

However, the syntax and type restrictions imposed on rewrite rules in these systems have prevented the design of completion procedures for higher-order rewrite systems (Nipkow and Prehofer, 1998). Nominal terms enjoy many useful properties, for instance, unification modulo α-equivalence is decidable and unitary (Urban et al., 2004) and nominal matching is linear (Calvès and Fernández, 2008a). Nominal rewriting can be implemented efficiently if we use closed rules, roughly speaking, closed rules do not contain unabstracted atoms and preserve the binding status of atoms during reduction - a natural restriction which is also imposed on Combinatory Reduction Systems by definition as well as other well-known higher-order formalisms (e.g. HRSs (Nipkow, 1991) and ERSs (Glauert et al., 2005; Khasidashvili, 1990)). Additionally, a completion algorithm for Nominal Rewrite Systems has been designed by Fernández and Rubio in (Fernández and Rubio, 2012). Therefore a practical interest arises, that of the possibility of transferring results between nominal rewriting and higher-order term rewriting by reason of the existing translations between Combinatory Reduction Systems and other higher-order rewriting formalisms. In particular, techniques and properties of the rewriting relation such as termination can be exported from nominal rewriting to Combinatory Reduction Systems.

This thesis first defines a pair of translation functions from Combinatory Reduction Systems to Nominal Rewrite systems and vice versa where the rewrite relation is preserved and (almost) reflected. The translation from Combinatory Reduction Systems to Nominal Rewrite systems is based on the translation given in (Fernández et al., 2004) where some issues have been corrected from the original translation as well as improving the naturality of the encoding by using closed rewriting, a formal treatment of the Barendregt variable convention in rule induction. An extension of nominal terms is then studied, where capture-avoiding substitution of atoms by terms is implemented in the syntax. Properties of the extended theory and an algorithmic representation of α-equality are then derived, providing a framework in which the notion of unification of extended nominal terms is defined, along with a proof of undecidability constructed by implementing an effective method to reduce *Hilbert's tenth problem* to unification of nominal terms extended with atom substitution. Two matching algorithms are then specified, a general one that finds the set of all possible correct solutions and only correct solutions to any (unifiable) matching problem by imposing a syntactic restriction on the matching algorithm to avoid instantiation of variables on one side of the matching constraint, and a unitary *simple-matching algorithm* which enjoys the property of unique most general solutions for the class of *simple matching constraints*. Both matching algorithms are then applied to the extension of Nominal Rewrite Systems we describe, providing an operational definition of closedness for extended nominal terms and, in the case of the *simple-matching algorithm*, finding a match during the rewriting process.

Finally, we use the framework of extended nominal rewriting to provide a *one-step reduction preserving translation* from Combinatory Reduction Systems to *standard extended Nominal Rewrite Systems* as well as a reduction-preserving translation from the class of standard extended Nominal Rewrite Systems to Nominal Rewrite Systems.

Contents

Introduction

This thesis is concerned with the relationship between two distinct research areas extending *term rewrite systems* with binding support, that of *higher-order rewriting formalisms*, where *Combinatory Reduction Systems* has been chosen as a representative, and that of *nominal rewriting formalisms*.

Term Rewrite Systems Term rewrite systems (Baader and Nipkow, 1998; Dershowitz and Jouannaud, 1990) (TRSs) play an important role in many areas of computer science providing a framework to reason about logic and computation. The theory is simple, syntax trees (also known as *terms*) built over a set of *variables* and a fixed set of *constant* and *function symbols* are transformed according to a set of *rewrite rules*. Take for instance the equational theory representing addition of natural numbers using Peano arithmetic where $\mathtt{0}$ is the constant zero, $\mathtt{suc}$ is a unary function symbol denoting the successor of a number and $\mathtt{plus}$ is the binary function symbol representing addition in prefix notation. Then, natural numbers can be expressed by terms over $\mathtt{0}$ and $\mathtt{suc}$ (e.g., $\mathtt{0} = \mathtt{zero}, \mathtt{suc}(\mathtt{0}) = \mathtt{1}, \mathtt{suc}(\mathtt{suc}(\mathtt{0})) = \mathtt{2}$ and so on) and the set of directed equations

$$\begin{array}{lll} \mathtt{plus}(\mathtt{0},\mathtt{X}) & \to_{zero} & \mathtt{X} \\ \mathtt{plus}(\mathtt{suc}(\mathtt{Y}),\mathtt{X}) & \to_{plus} & \mathtt{suc}(\mathtt{plus}(\mathtt{Y},\mathtt{X})) \end{array}$$

represents the addition algorithm as a TRS such that a computation to resolve $\mathtt{2}+\mathtt{1}$ is executed by application of equations from left to right over the term representing $\mathtt{2}+\mathtt{1}$ as follows,
$\mathtt{plus}(\mathtt{suc}(\mathtt{suc}(\mathtt{0})),\mathtt{suc}(\mathtt{0})) \to_{plus} \mathtt{suc}(\mathtt{plus}(\mathtt{suc}(\mathtt{0}),\mathtt{suc}(\mathtt{0})))$
$\to_{plus} \mathtt{suc}(\mathtt{suc}(\mathtt{plus}(\mathtt{0},\mathtt{suc}(\mathtt{0})))) \to_{zero} \mathtt{suc}(\mathtt{suc}(\mathtt{suc}(\mathtt{0})))$.

This is known as *term rewriting* and the irreducible term $\mathtt{suc}(\mathtt{suc}(\mathtt{suc}(\mathtt{0})))$ is the result of the computation, that is, its *normal form*. TRSs have many applications such as in specification, analysis and verification of programs, security, compiler building, automated deduction, artificial intelligence and many other fields. As a result, there is a large body of research documenting the properties of TRSs (see (Baader and Nipkow, 1998; Dershowitz

and Jouannaud, 1990) for a survey). The main topics of research in TRSs can be divided into two categories, *termination* and *confluence*.

Termination is a highly desirable property of logical systems, concerned on whether rewriting sequences can be assumed to be finite, that is, does the `plus` algorithm terminate after finitely many steps for any term in the set of Peano natural numbers? The answer is yes, `plus`-example is terminating. However, one could add the rewrite rule $\texttt{plus}(X,Y) \rightarrow_{swap} \texttt{plus}(Y,X)$ and then the algorithm is no longer terminating: $\texttt{plus}(\texttt{suc}(\texttt{suc}(0)),\texttt{suc}(0)) \rightarrow_{swap} \texttt{plus}(\texttt{suc}(0),\texttt{suc}(\texttt{suc}(0))) \rightarrow_{swap} \texttt{plus}(\texttt{suc}(\texttt{suc}(0)),\texttt{suc}(0)) \cdots$. Termination of term rewrite systems has been extensively studied and several powerful methods for establishing termination are available (e.g. (Arts and Giesl, 2000; Dershowitz, 1987; Steinbach, 1994)).

Confluence can be seen as a study of non-determinism. Term rewriting is inherently non-deterministic because a term can contain more than one *reducible expression* (also known as *redex*). For instance, $\texttt{plus}(0,\texttt{suc}(\texttt{plus}(0,\texttt{suc}(0))))$ rewrites in one step to both $\texttt{suc}(\texttt{plus}(0,\texttt{suc}(0)))$ and $\texttt{plus}(0,\texttt{suc}(\texttt{suc}(0)))$. However, the result of rewriting a term with respect to a TRS can still be uniquely determined provided the multiple reducible expressions do not interfere with each other. This is the case for our example where both terms rewrite in one step to normal form $\texttt{suc}(\texttt{suc}(0))$. Similarly to termination techniques, confluence techniques are widely available (e.g. (Huet, 1980; Knuth and Bendix, 1970)). To summarise, a TRS R provides a decision procedure for an equational theory E if R represents the same equational theory as E, is finite, terminating and confluent. Following this four conditions one can conclude that the `plus`-example algorithm is thus a decision procedure for the equational theory representing addition of natural numbers using Peano logic and therefore it can be used to determine equality among the terms in the set built over `0` and `suc` (and variables). This leads to an interesting topic related to both termination and confluence: how can one make an *incomplete* equational theory complete so that it provides a decision procedure?, that is, if the logical system is incompletely specified, how can one extend it in a suitable way so that it preserves the four conditions outlined above? This is known as the *completion* procedure and the central idea is to limit attention to certain *critical* deductions obtained from overlappings of left-hand side (LHS) of rules. Then, these overlappings are used to generate new rules which are sound with respect to the theory and where the right-hand side (RHS) of the rule is a normal form for the current system and the LHS of the rule cannot be rewritten by other existing rules. Knuth and Bendix designed the completion procedure in their seminal paper (Knuth and Bendix, 1970). For a survey, we direct the reader to (Dershowitz, 1989).

Extensions of Term Rewrite Systems with Binding Support As we have shown, TRSs do well with first-order objects like numbers or lists, for example, but cannot accommodate higher-order functions, that is, functions that take other functions as arguments. Take for instance the well-known higher-order function `map` which applies a function to all the elements of a list,

$$\begin{array}{lcl} \texttt{map(f,nil)} & \rightarrow & \texttt{nil}, \\ \texttt{map(f,cons(x,xs))} & \rightarrow & \texttt{cons(f(x),map(f,xs))}. \end{array}$$

Notice that in the second rule `f` is both a variable and a unary function symbol, that is, `f` is a *function pointer* in the rule. Although this is a legal functional program, it is not a legal TRS since `f` cannot appear both as a function and a variable.

Another issue is term rewriting with bound variables or *higher-order rewriting* as it is called nowadays. Manipulating terms with bound variables is outside the scope of TRSs, a simple formula like

$$\forall x(\exists y P(x,y)) \vee \exists x(\forall y Q(x,y))$$

cannot be represented by TRSs because substitution is a simple replacement of variables by terms and the variables on the right-hand side of the disjunction symbol range over a distinct domain of discourse than the variables on the left. A more complex form of substitution is needed to deal with variable substitution in binding contexts.

A classic solution to both these problems is extending TRSs with the λ-calculus (Barendregt, 1984) a rewriting formalism introduced by Alonzo Church in the 1930s which offers a simple semantics to formally represent *computable functions* and therefore providing variable binding and higher-order substitution for free.

In the λ-calculus terms are built from a set $\mathscr{V}$ of variables using λ-abstraction and application as shown in the following grammar:

$$s,t ::= x \mid \lambda x.s \mid (s\ t) \quad (\text{where } x \in \mathscr{V})$$

In a λ-term $\lambda x.s$, the construct $\lambda x.$ *binds* all occurrences of variable x in s. The set of *free* variables in a λ-term consists of the set of variables which are not bound in the λ-term by symbol λ.

In the calculus, the semantics of λ-terms is defined by equality with respect to three equational laws denoted by α, β and η (we do not expand on the η-rule since it is not required).

α-equality (also called α-conversion) defines λ-terms to be syntactically equivalent modulo renaming of their bound variables. Then, the pair of λ-terms $\lambda x.\lambda y.xy$ and $\lambda z.\lambda x.zx$

represents the same λ-term, so now one writes $\lambda x.\lambda y.xy \equiv \lambda z.\lambda x.zx$ which shows that we are not working with syntax trees as in TRSs but with α-equivalence classes. Accordingly, one can always assume names in a binder to be *fresh*. This freshness assumption is often expressed by means of an informal side condition denoting that the chosen name of the binder does not appear as a free variable inside a term. In fact, following Barendregt's variable convention from his seminal book (Barendregt, 1984) one often assumes bound variables to be distinct from free variables across all the terms in the context one is working on (for instance some definition or proof).

An application of a function $\lambda x.s$ to a term t, $((\lambda x.s)t)$, is just notation, that is, the function is not yet evaluated. β-equality captures the notion of function evaluation and it is defined in terms of a capture-avoiding substitution, $s\{x \mapsto t\}$, that is, the process of replacing all free occurrences of a variable x in s by λ-term t, choosing a representative of s whose bound variables are neither in the domain nor in the image of $\{x \mapsto t\}$. Therefore evaluation of a function must be performed explicitly using the β-rule. This intricate substitution process involves the use of more informal freshness side conditions too. Higher-order rewriting systems benefit from this approach by performing rewriting modulo the *substitution calculus* either in its untyped form (as for instance Combinatory Reduction Systems) or more commonly as a typed calculus since, unlike untyped λ-calculus, β-reduction is guaranteed to terminate (Tait, 1967).

There is a lack of a standard higher-order formalism, in fact, a large body of distinct higher-order formalisms combining TRSs with the λ-calculus have been proposed to study particular properties of rewriting (e.g. termination, confluence) or to deal with weaknesses of previous higher-order formalisms. Since the aim of this thesis is to compare Combinatory Reduction Systems and Nominal Rewriting Systems, we select *not* to discuss other higher-order formalisms exhaustively but to enumerate chronologically those that are historically most important and give references to the literature and an overview of the similarities instead. Not only is overkill to provide a summary of all the available higher-order rewriting formalisms but also the relationship among them (and with Nominal Rewrite Systems in (Fernández and Gabbay, 2007)) has already been thoroughly studied by other fellow researchers as we show both in the list and more extensively in the related work chapter (see Chapter 8).

In historical order we have:

- CS = Contraction Scheme. Introduced by Aczel in 1978 (Aczel, 1978).

- CRS = Combinatory Reduction System. Introduced by Klop in his Phd thesis in 1980 (Klop, 1980). An alternative definition has been given in (Klop et al., 1993) and has since become the standard form of CRS.

- HOTRS = Higher-Order Term Rewriting System. Introduced by Wolfram in his Phd thesis (see (Wolfram, 1993)).
- ERS = Expression Reduction System. Introduced by Khasidashvili (Khasidashvili, 1990) (and later revised in (Glauert et al., 2005)).
- PHRS = Pattern Higher-order Rewrite System. Introduced by Nipkow in (Nipkow, 1991).
- AFS = Algebraic Functional System. Introduced by Jouannaud and Okada in (Jouannaud and Okada, 1991).
- IS = Interaction System. Introduced by Asperti and Lavene in (Asperti and Laneve, 1993).
- CLC = Conditional λ-Calculus. Introduced by Takahashi in (Takahashi, 1993).
- HORS = Higher-Order Rewrite Systems. Introduced by van Oostrom and van Raamsdonk in (van Oostrom, 1994) as an unifying framework for the already existing formalisms (including CRS) by separating the notion of *substitution calculus* (e.g. untyped λ-calculus in CRS, typed λ-calculus in HRS) from the notion of terms.
- HRS = Higher-Order Rewrite System. Following Wolfram (Wolfram, 1993), the pattern restriction on rules from PHRS was dropped by Mayr and Nipkow in (Mayr and Nipkow, 1998).
- CRSX = Combinatory Reduction System with eXtensions. Introduced by Rose in his Phd thesis in (Rose, 1996) as a rewriting formalism for business applications. It has seen major changes in recent years (Rose, 2011)
- ADTS = Abstract Data Type System. Introduced by Jouannaud and Okada in (Jouannaud and Okada, 1997).
- ρ-calculus = Rewriting Calculus. Introduced by Cirstea and Kirchner in (Cirstea and Kirchner, 1998).
- IDTS = Inductive Data Type System. Introduced by Blanqui in (Blanqui, 2006).
- AFSM = Algebraic Functional System with Meta-variables. Introduced by Kop in her Phd thesis in (Kop, 2012). Similarly to HORS, Kop provided a unifying framework for the study of higher-order termination in many of the aforementioned formalisms (including CRS)

These higher-order rewriting formalisms share the following concepts among them:

- A distinction between variables and meta-variables. Meta-variables are first-order thus cannot be bound; they are the variables of the algebra underlying the term rewrite systems. Then, variables are the higher-order variables that can be bound and substituted by terms to represent substitution of object-level variables.

- Each system offers at least one symbol or construct for abstraction. For instance CRSs has λ from λ-calculus at the meta-level and [-]- at the level of the language. ERS allows the user to define its own binders providing a more natural representation (although it is syntactic sugar in that ERSs can be expressed within a syntax having only one binder symbol (Glauert et al., 2005)).

- There is a distinction between meta-terms (contains variables and meta-variables) and terms (meta-terms containing no meta-variables), rewrite rules are composed by meta-terms whereas the rewrite relation is induced by terms.

- The rewriting relation is generated from rewriting rules by means of assignments and it may involve some substitution steps.

- They have a substitution calculus closely related to the λ-calculus, that means α-conversion cannot be avoided.

Combinatory Reduction System The CRS introduced by Klop is considered the first framework combining TRS and the λ-calculus. As such, the CRS framework is viewed as a benchmark where emerging higher-order rewriting formalisms analyse their derivation space by comparing their expressive power against that of the CRS. The meta-language of CRS, that is, the language in which the notions of rewrite and substitution are expressed are based on the untyped λ-calculus and higher-order matching, that is, matching modulo the equational laws of the λ-calculus. Higher-order matching (a particular instance of unification where one of the λ-terms is closed) and, consequently, full higher-order unification are undecidable (Goldfarb, 1981; Huet, 1973; Levy, 1998; Levy and Veanes, 2000). As a consequence, Klop restricted the structure of rule patterns by not allowing occurrences of free variables and making the correspondence between meta-variables and variables explicit, so that the most primitive operation of rewriting, that is, searching for a redex, was then decidable. Nevertheless, higher-order unification for simply-typed λ-terms was conjectured to be decidable by Huet (Huet, 1975) and Miller identified, roughly ten years after CRS was published, a well-behaved special case of *higher-order patterns* (Miller, 1991a) (similar to CRS rule patterns) for which unification is decidable and unitary and which re-ignited

the interest on higher-order rewriting formalisms, starting with PHRS by Nipkow (Nipkow, 1991).

Klop was interested in generalising the λ-calculus theory and more precisely the confluence and finite developments theory. The finite development theory gives sense to parallel reductions in the λ-calculus by stating that one can rewrite any given set of reducible expressions in a λ-term without looping or being concerned about the order in which these redexes are rewritten. This theorem can be used to prove confluence, thus ensuring determinism of rewrites and uniqueness of normal forms (Lévy, 2007). Since Klop was not interested on termination and he could not use a syntactic search based on first-order matching (because of the λ-calculus), he chose to search via finite developments so that the search was finite in the presence of β-rewrites (and because of the absence of typing assumptions) (Jouannaud, 2005).

Meta-variables in CRS are labelled with a fixed arity which must be preserved across the distinct occurrences that may occur in a CRS meta-term. The dependence of meta-variables on names (of binders) is given explicitly in CRSs, for instance, writing $X^1(a)$ means that meta-variable X^1 can depend only on one name, in this case a, as given by the arity of X^1. Accordingly, meta-terms in CRS rewrite rules have to be closed (that is, no free variable occurrences) so that if $[a]X^1(a)$ is the LHS meta-term of a CRS rule then X^1 denotes a unary function pointer.

Extracting the rewrite relation from the CRS rewrite rules is not straight-forward since one has to deal with the λ-calculus at the meta-level. A *valuation* is a binding from meta-variables to *substitutes* which are the $\underline{\lambda}$-terms (λ underlined to denote that is *based on* λ-calculus) for which the meta-variables are replaced, in a syntactic manner. Substitutes have the form $\underline{\lambda}(x_1,\ldots,x_n).t$ where the tuple $(x_1,\ldots,x_n)$ contains the variables bound in t and the arity of the tuple has to be the same as the arity of the meta-variable that replaces in order to induce a substitution. Safety conditions are instrumental to avoid free variables x in the substitute being captured by abstractors $[x]$ at the level of the CRS language. This is basically achieved informally by means of the Barendregt convention, that is, rename all bound variables in both the rules and the valuations so that all bound variables are distinct among themselves and from the free variables. Repeat after each rewrite step. When a valuation is applied, additional substitution computations may be needed to evaluate a function as we have already discussed when describing the λ-calculus. However, this is done at the meta-level and therefore not seen as part of the rewrite relation.

We conclude with CRS by defining the higher-order function map from the beginning of the section in CRS notation,

$$\begin{array}{lcl} \mathtt{map}([\mathtt{a}]\mathtt{F}^1(\mathtt{a}),\mathtt{nil}) & \to & \mathtt{nil}, \\ \mathtt{map}([\mathtt{a}]\mathtt{F}^1(\mathtt{a}),\mathtt{cons}(\mathtt{X}^0,\mathtt{XS}^0)) & \to & \mathtt{cons}(\mathtt{F}^1(\mathtt{X}^0),\mathtt{map}([\mathtt{c}]\mathtt{F}^1(\mathtt{c}),\mathtt{XS}^0)). \end{array}$$

As we previously mentioned above, due to the untyped nature of the underlying logic for the substitution calculus, this CRS is non-terminating. Take for instance the CRS term $\omega = \mathtt{map}(\Omega,\mathtt{cons}(\Omega,\mathtt{nil}))$ taken from Kop's thesis (Kop, 2012, Section 3.7.1) where $\Omega = [\mathtt{x}]\mathtt{map}(\mathtt{x},\mathtt{cons}(\mathtt{x},\mathtt{nil}))$. Using the second rule, ω rewrites to $\mathtt{cons}(\mathtt{map}(\Omega,\mathtt{cons}(\Omega,\mathtt{nil})),\mathtt{map}(\Omega,\mathtt{nil})) = \mathtt{cons}(\omega,\mathtt{map}(\Omega,\mathtt{nil}))$ which contains the original term.

Nominal rewrite formalisms A more recent solution to the problem of variable binding in TRSs can be found in Nominal Rewrite Systems (Fernández and Gabbay, 2007; Fernández et al., 2004) (NRSs) in which names and name binding are encoded using built-in data structures called atoms and abstraction terms. NRS is more complex than TRS yet simpler than higher-order rewriting formalisms. The underlying idea of NRS is based on the seminal work by Gabbay and Pitts on *nominal sets* (Gabbay and Pitts, 2002) as well as the work on *nominal unification* done by Urban, Pitts and Gabbay in (Urban et al., 2004) and Pitts in *nominal logic* (Pitts, 2003). *Nominal terms* have just enough structure to capture α-equality while still admitting a first-order semantics and a decidable equality theory and unification problems.

The main components of the nominal approach to rewriting are:

- a set of infinite syntactic names $a,b,c\ldots \in \mathscr{A}$ called *atoms*,
- a swapping operation $(a\ b)\cdot$- that swaps two atoms within a term,
- an abstraction operation $[a]s$ that binds an atom within an expression and admits equality up to α-equivalence,
- a freshness relation $a\#s$ that holds between an atom and a term when the term is independent from the atom, that is, the atom does not occur free in the term, and
- a decidable unification algorithm for nominal terms which is derived by simple recursion on the syntactic structure of nominal terms and where freshness assumptions are taken into account formally, by means of the freshness relation.

The intuition behind nominal techniques is that α-equality can be defined in terms of swappings and the freshness relation. In particular, equality for a pair of abstraction terms

$[a]s \approx_\alpha [b]t$ can be found by checking whether the body of the abstractions can be made equal by swapping their bound atoms and the atom bound on one side does not appear free in the other.

Additionally, Gabbay and Pitts identified the following principles for nominal logic which affect nominal rewriting:

- Relations are invariant up to swapping, that is, the choice of particular atoms is irrelevant, this is known as *equivariance*,
- there always can be found a fresh atom for any term.

Similarly to CRSs, in NRSs there is a distinction between meta-variables and object-level variables, in this case, meta-variables are called *variables* or *unknowns* and are first-order in that they cannot be bound and substitution is a simple replacement whereas object-level variables are concrete data structures called atoms and can be bound. Unlike CRSs, dependencies between variables and atoms is left in this case implicit and thus variables have arity zero. α-conversion is obtained by means of a small built-in logic derived over a syntactic nominal structure. As a result, the notion of rewriting on nominal terms behaves as first-order rewriting but uses matching modulo α-conversion and syntactic substitution, without the need to introduce meta-substitutions or β-rewrites as it is common in the substitution calculus of higher-order rewriting formalisms. Accordingly, the notion of function evaluation and higher-order substitution is therefore added to NRSs as a set of rewrite rules.

The rewrite relation in NRSs is not necessarily induced on closed rules with meta-variables as in CRSs, neither the matched term has to be ground (no variables). NRSs are very expressive and allow rewrite rules to contain free atoms and/or no variables. However, due to equivariance, selecting a rewrite rule that matches a given term is an NP problem in general (Cheney, 2004a, 2010) since unabstracted atoms in the rule also need to be matched. However, Gabbay and Fernandez found in (Fernández et al., 2004) that closed rules can avoid the exponential cost. Nominal matching is linear in this case, as found by Calvès and Fernández in (Calvès and Fernández, 2008b).

Because of the good properties of nominal terms, many techniques have been able to be generalised from TRS to NRSs, most notably an extension of the recursive path ordering to check termination of TRSs and completion procedure for NRS both by Fernández and Rubio in (Fernández and Rubio, 2012). Completion procedures for higher-order rewriting formalisms are very difficult to define because of the use of β-reduction and the restrictions on the structure of rewrite rules, as explained in (Nipkow and Prehofer, 1998).

We conclude with NRS by defining a rewrite system for the higher-order function map from the beginning of the section,

$$\begin{array}{lcl} \mathtt{map}(\mathtt{lam}([\mathtt{a}]\mathtt{F}),\mathtt{nil}) & \to & \mathtt{nil}, \\ \mathtt{map}(\mathtt{lam}([\mathtt{a}]\mathtt{F}),\mathtt{cons}(\mathtt{X},\mathtt{XS})) & \to & \mathtt{cons}(\mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{F}),\mathtt{X}),\mathtt{map}(\mathtt{lam}([\mathtt{a}]\mathtt{F}),\mathtt{XS})). \\ \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{X}),\mathtt{Y}) & \to & \mathtt{X}[\mathtt{a} \mapsto \mathtt{Y}]. \end{array}$$

where the bottom rewrite rule represents the β-rewrite rule from the λ-calculus and notation $t[a \mapsto s]$ is syntactic sugar for $\mathtt{sub}([a]t,s)$. The binary function symbol sub represents the higher-order substitution of the λ-calculus and its behaviour needs to be given meaning by additional rewrite rules which we add below.

$$\begin{array}{rcll} \mathtt{a}[\mathtt{a} \mapsto \mathtt{X}] & \to & \mathtt{X} & \\ \mathtt{a}\#\mathtt{Y} \vdash \mathtt{Y}[\mathtt{a} \mapsto \mathtt{X}] & \to & \mathtt{Y} & \\ (\mathtt{f}\,\mathtt{X})[\mathtt{a} \mapsto \mathtt{Y}] & \to & \mathtt{f}\mathtt{X}[\mathtt{a} \mapsto \mathtt{Y}] & \text{for each } \mathtt{f} \text{ in } \Sigma \\ (\mathtt{X}_1,\ldots,\mathtt{X}_\mathtt{n})[\mathtt{a} \mapsto \mathtt{Y}] & \to & (\mathtt{X}_1[\mathtt{a} \mapsto \mathtt{Y}],\ldots,\mathtt{X}_\mathtt{n}[\mathtt{a} \mapsto \mathtt{Y}]) & \\ \mathtt{b}\#\mathtt{Y} \vdash ([\mathtt{b}]\mathtt{X})[\mathtt{a} \mapsto \mathtt{Y}] & \to & [\mathtt{b}](\mathtt{X}[\mathtt{a} \mapsto \mathtt{Y}]) & \end{array}$$

with Σ being the set of function symbols in the NRS above along with their fixed arity. The syntax of nominal terms can be found in Chapter 1.

Using the same example than in the CRS for map we have the NRS term $\omega_{\mathtt{NRS}} = \mathtt{map}(\Omega,\mathtt{cons}(\Omega,\mathtt{nil}))$ where $\Omega = \mathtt{lam}([\mathtt{a}]\mathtt{map}(\mathtt{a},\mathtt{cons}(\mathtt{a},\mathtt{nil})))$ and which rewrites to $\mathtt{cons}(\mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{map}(\mathtt{a},\mathtt{cons}(\mathtt{a},\mathtt{nil}))),\Omega),\mathtt{map}(\Omega,\mathtt{nil}))$ using the second rewrite rule for function symbol map and, again, applying the rule that represents the β-rewrite rule we have $\mathtt{cons}((\mathtt{map}(\mathtt{a},\mathtt{cons}(\mathtt{a},\mathtt{nil})))[\mathtt{a} \mapsto \Omega],\mathtt{map}(\Omega,\mathtt{nil}))$. Observe that the function evaluation that was implicitly done in the CRS meta-language needs to be computed in NRS by means of the additional rewrite rules for substitution so that there are roughly seven more rewrite steps to obtain the term $\mathtt{cons}(\omega,\mathtt{map}(\Omega,\mathtt{nil}))$. If the aim of this thesis is to compare NRSs and CRSs by building a translation between both formalisms which preserves the (one-step) rewrite relation, it is obvious that we need to address the fact that NRS systems do not have a built-in operation for capture-avoiding substitution. The need to incorporate such device to the nominal approach was already suggested by Cheney in (Cheney, 2004b) and Cheney and Urban in (Cheney and Urban, 2004, 2008) with respect to α-Prolog, an extension of Prolog using nominal abstract syntax and by Fernández and Gabbay in NRSs (Fernández and Gabbay, 2007), among others. Further, Fairweather, Fernández, Szasz and Tasistro extended the syntax of nominal terms to include substitution of atoms by terms in their work on dependent types in (Fairweather et al., 2015) with a view to define a *nominal logical framework* in the future. This shows that extending the framework of nominal rewriting

with a device for capture-avoiding substitution of atoms by terms is highly desirable, not only from a theoretical point of view but also from a practical one. We have accepted the challenge.

This thesis explores an alternative approach of rewriting with names, which is called *nominal rewriting with atom substitution* or *extended nominal rewriting* for short, in which substitution of atom by terms is no longer represented by a function symbol and a set of rules but encoded in the syntax using a built-in operator $[\cdot \mapsto \cdot]$ that simulates the behaviour of capture-avoiding substitution as defined for the λ-calculus. To define this new nominal rewriting framework we have

- shown that the behaviour of $[\cdot \mapsto \cdot]$ corresponds to that of capture-avoiding atoms substitution;
- given a logical presentation of α-equality for nominal terms with atom substitution (referred to as *extended nominal terms* from now on) and the theorems demonstrating the properties of the logical system, for instance that our presentation is indeed an α-equality on extended nominal terms;
- described an algorithm to check α-equality between extended nominal terms which we have proven sound and complete;
- proved that, similarly to second-order unification of lambda-terms (see for instance (Goldfarb, 1981; Levy and Veanes, 2000)) unification of extended nominal terms is undecidable;
- restricted the class of unification problems to matching problems in order to provide a unification algorithm that *correctly* produces *every possible principal solution* to a matching problem, if such problem is unifiable. Such matching algorithm can be used as an operational definition of closedness for extended nominal terms and rewrite rules;
- derived a unitary pattern matching algorithm for a subclass of matching problem;
- obtained a characterization of the class of extended nominal rewrite rules for which pattern matching provides at most one principal solution;
- extended the notion of rewrite relation for both general (where equivariance must be taken into account) and closed rules.
- shown the correspondence between extended nominal rewriting and (non-extended) nominal rewriting.

Motivation Nominal and higher-order rewrite formalisms provide a means for describing binding structures in TRSs. These approaches have created two different communities which use fundamentally distinct semantic foundations (i.e., λ-calculus in higher-order, nominal sets in NRSs) to deal with variable binding and α-conversion. Some of their main differences are summarised below.

The use of λ-calculus in higher-order rewriting formalisms provides a notion of binding and a substitution calculus for free. Then, rewriting is performed modulo the equality laws of λ-calculus. However, this also increases the complexity of the matching process since a matching algorithm may have to handle more complex forms than α-equality, for instance β-equality where an application function is equated to its normal form. There is a large body of work describing techniques to prove confluence and termination (e.g. (Blanqui, 2006; Kop, 2012; van Oostrom, 1994; van Raamsdonk, 1996)) in higher-order rewriting formalisms, however due to restrictions on structure and/or type imposed on higher-order patterns, it has not been possible to design completion algorithms where the imposed restrictions are preserved (Nipkow and Prehofer, 1998) (for the general case) after application of the algorithm. The rewrite relation in higher-order rewriting is induced on terms, where meta-terms are used to describe rewrite rules.

On the other hand, nominal rewrite systems include such novelty devices as variables that can be abstracted but behave like constants therefore allowing manipulation, an atom swapping operation, a freshness relation and an abstraction operation for atom-binding. However, abstractions are not considered to be functions as in λ-calculus therefore variables may mention arbitrary atoms which are then captured by the abstraction after an instantiation. Also, there is no built-in notion of function evaluation and therefore it must be added to systems along with a set of rules for capture-avoiding atom substitution. Matching is performed modulo α-equivalence instead of the usual α- β-equivalence of λ-calculus. α-equality is elegantly handled by a small built-in logic by means of the swapping operation and the freshness relation. Rewrite rules are considered equivalent up to renaming of both variables and atoms, be they abstracted or unabstracted. Rewrite rules containing only variables are allowed and the rewrite relation is also induced on terms with variables.

Despite these differences, both formalisms are closely related, using distinct techniques to induce a rewrite relation in rewriting systems with binding support.

Interest in establishing a relation between the nominal and the higher-order approach has already been shown with respect to pattern unification (Cheney, 2005; Levy and Villaret, 2012) and abstract syntax specifications (Gacek, 2010). This demonstrates that there is a niche for studying the relationship between both rewriting formalisms. So, why CRS?

Because CRS was the first rewriting formalism to combine TRSs and λ-calculus and therefore it is often used as a benchmark for expressiveness for other emerging rewriting formalisms. As a result, there is a theoretical interest on providing a relationship between NRS and CRS because the expressive power of nominal and higher-order rewriting formalisms is put in evidence. Moreover, since there has already been established an association between other higher-order rewriting formalisms with CRS, a practical benefit arises, that of the possibility of transferring results between both rewriting communities. In particular, techniques and properties of the rewriting relation such as termination, can be exported from NRS to CRS. Additionally, the notation of both CRS and NRS is very similar since λ-calculus is relegated to the meta-level, this and the fact that CRS is untyped makes the transition from one system to the other much simpler.

I have argued that the study of the relationship between higher-order and nominal rewriting formalisms and more particularly between that of CRS and nominal rewriting has an specific appeal for the research community. Now, I demonstrate how we built such relationship between formalisms.

Outline

My thesis is that

> *Unification of nominal terms with atom substitution is undecidable. Nevertheless, the matching process is both decidable and unitary for a particular class of extended terms thus providing a means to mechanise the rewriting process for the extended nominal framework. The extension of nominal rewriting is then used to demonstrate there is a direct correspondence between Combinatory Reduction Systems and extended Nominal Rewrite Systems.*

To support this thesis, I shall provide a translation function from CRSs to extended NRSs and a translation function from a class of *standard* NRSs to CRSs. Both translation functions preserve the one-step rewrite relation (therefore there is a direct correspondence between both formalisms). This can be found in Chapter 7 along with another interesting result, the correspondence between extended NRSs (eNRSs) and NRSs by means of an encoding of a class of extended nominal rules and terms to non-extended rules and terms.

This is not the only original contribution of this thesis. I begin by providing a rewrite-preserving translation function from a class of *standard* NRSs to CRSs. This work has already been published in (Domínguez and Fernández, 2014) and an implementation can be found in (Domínguez, 2014). Then, I continue by defining another translation function,

this time encoding CRSs in NRSs. The function is based on the same translation function previously published in (Fernández et al., 2004). I have corrected and improved the original function by adding a more fine-grained auxiliary function to build swappings and also by using *closed rewriting* (see *Summary of published work* in Chapter 1.4 for more details). It is shown that the rewrite relation is preserved for this case when adding a set of rewrite rules to simulate the substitution calculus of λ-calculus. This material has also been published, in this case in the LMCS journal (Domínguez and Fernández, 2015), along with the previous material and the complete set of proofs. Both translation functions and their properties will be applied once again in Chapter 7 when providing a translation between CRSs and extended NRSs. *Although both translation functions are contributions to this thesis, I have decided not to include them fully in this thesis since they have already been published.* However, I have included a summary of the published work after the preliminaries chapter (see Chapter 1) for the sake of clarity when reading Chapter 7.

Another contribution is in Chapter 4, where I demonstrate that unification of nominal terms extended with atom substitution is undecidable by designing an effective method to reduce *Hilbert's tenth problem* to nominal unification of extended terms. Nevertheles, In Chapter 5 I identify a class of problems for which unification is decidable. Such class of decidable unification problems is obtained by imposing a syntactic restriction on the set of variables which can be instantiated, in order to convert the unification algorithm into a matching one. Then, I proceed to define a unification algorithm that produces the complete set of solutions to any (unifiable) unification problem within such class and demonstrate its correctness. The unification algorithm for matching problems is done with a view to provide a mechanism for matching and an operational definition of closedness for extended terms when generalising nominal rewriting to the extended framework in Chapter 6. Additionally in Chapter 5, I also characterise a class of matching constraints, which I call *simple*, that have at most one principal solution. Then, I reduce the unification algorithm for matching problems to provide a unitary matching algorithm. This is done by applying further restrictions and strategies to enforce unitary solutions when applying the algorithm to simple extended nominal terms. Correctness results for the simple-matching algorithm are given, thus providing the necessary tools to automate rewriting on Chapter 6. Finally, my last contribution is in Chapter 6 where I extend the notions of *elementary rewriting* (as named in (Fernández et al., 2004) to refer to general nominal rewriting) and closed rewriting from NRS to the extended framework so that I can build the translation functions that will support my thesis.

Overview of the thesis The rest of the thesis is organised as follows. In Chapter 1 we recall the formalisms of TRS, NRS and CRS. Then, we offer a summary of our published

work as a reference when reading Chapter 7. In Chapter 2 we introduce the syntax of nominal terms extended with atom substitution, provide a logical presentation of the freshness and α-equality relation for extended terms and prove some fundamental properties about extended terms. In Chapter 3, we transform the logical presentation of both the freshness and the α-equality relation into a sound and complete constraint checking algorithm which it will be used to build a unification algorithm on top. In Chapter 4, we define unification and pattern unification problems and provide a proof of undecidabillity for unification of extended terms by finding an effective method of reducing Hilbert's tenth problem to extended nominal unification. The chapter is concluded by outlining the issues surrounding decidability of unification for a restricted class of unification problems and the generation of unique most general solutions. A solution to such issue is postponed until the following chapter. In Chapter 5, we define matching and pattern matching problems, find a class of matching constraints which has at most one principal solution and introduce two matching algorithms, a general one restricting variable instantiation to one side and a *simple matching algorithm* which adds further restrictions to ensure unicity of results when applied to simple matching constraints. Additionally, we provide examples and also prove correctness of the matching algorithms. In Chapter 6, we define a suitable theory for nominal rewriting in the presence of atom substitutions which extends the theory from NRS, providing definitions for elementary and closed rewriting. In Chapter 7, we provide a pair of translation functions from a class of eNRSs to CRSs and back again that preserves the rewrite relation and illustrate their application with some interesting examples. Also, we give a definition to reduce a class of extended nominal rules and terms to their non-extended analogues. In Chapter 8, we provide a reference to work related to ours and in Chapter 9 we provide a final word on conclusions and future work.

Part I

Preliminaries and Summary of Our Published Work

Chapter 1

Preliminaries

In this chapter we briefly recall the syntax and main concepts of term rewriting systems following the reference book from Baader and Nipkow in (Baader and Nipkow, 1998). Then, we provide two representations of term rewrite systems with variable binding with distinct semantic foundations: nominal rewriting (Fernández and Gabbay, 2007; Fernández et al., 2004) which follows the nominal approach and Combinatory Reduction Systems (Klop, 1980; Klop et al., 1993) (we follow the notation given in (Klop et al., 1993)), a well-established representative of higher-order rewriting formalisms which uses λ-calculus in the meta-language as a substitution calculus.

1.1 Term Rewrite Systems

Definition 1.1.1. A signature Σ is a set of function symbols where each $f \in \Sigma$ is associated with a non-negative integer n, the **arity** of f. Function symbols of zero arity are also called **constant symbols**.

Fix a countably infinite set of **variables** $\mathscr{X}$ ranged over $X, Y, Z, \ldots$ and assume that Σ and $\mathscr{X}$ are disjoint.

Definition 1.1.2 (First-order term grammar)**.** The set of first-order terms are generated by the following grammar

$$s, t ::= X \mid f^n(s_1, \ldots, s_n)$$

Example 1.1.3 (First-order terms)**.** The following examples of first-order terms are generated following the function symbols representing addition in Peano arithmetic: $\Sigma =$

$\{0^0, \mathtt{suc}^1, \mathtt{plus}^2\}$.

$$0 \quad \mathtt{suc}(0) \quad \mathtt{suc}(\mathtt{suc}(0)) \quad \mathtt{suc}(\mathtt{suc}(\mathtt{suc}(0)))$$
$$\mathtt{plus}(0,0) \quad \mathtt{plus}(\mathtt{suc}(0),\mathtt{suc}(\mathtt{suc}(0))) \quad \mathtt{plus}(\mathtt{plus}(0,\mathtt{suc}(0)),0)$$

Let $\mathtt{vars}(t)$ be the set of variables occurring in t and use $\equiv$ to denote syntactic equality over terms.

Definition 1.1.4. A **substitution** σ is a finite mapping from variables to terms. The application of a variable σ to a term t is written as $t\sigma$ and represents the simultaneous instantiation of the variables in t, $\mathtt{vars}(t)$, which occur in the domain of σ.

Definition 1.1.5. A **matching problem** asks for any pair of terms l,t such that $\mathtt{vars}(l) \cap \mathtt{vars}(t) = \varnothing$ whether there exists a substitution σ such that $l\sigma \equiv t$.

Write $C[t]$ to represent some term C in which at a specific position of its syntax tree, occurs the sub-term t.

Definition 1.1.6. A **rewrite rule** R is a binary relation $\rightarrow$ on a pair of terms (l,r), written $l \rightarrow r$, such that $\mathtt{vars}(r) \subseteq \mathtt{vars}(l)$.

A **term rewriting system**, $\mathscr{R}$, is a set of rewrite rules over a particular signature Σ.

The **one-step rewrite relation** generated by a term rewriting system $\mathscr{R}$ is the least relation of tuples $s \rightarrow_{\mathscr{R}} t$ such that, for any rewrite rule $l \rightarrow r$ of $\mathscr{R}$ and any substitution σ then,

$$s \equiv C[s'], \quad l\sigma \equiv s' \text{ and } C[r\sigma] \equiv t$$

Write $s \rightarrow^* t$ for the reflexive, transitive closure of this one-step relation.

Informally, the one-step rewrite relation says that, if a subterm of a term matches the LHS of a rewrite rule then it can be rewritten to a corresponding instance of the RHS.

Example 1.1.7 (Term rewriting system)**.** Following the signature from Example 1.1.3 we can represent primitive recursive addition of natural numbers as follows

$$\begin{array}{lll} \mathtt{plus}(0,\mathtt{X}) & \rightarrow_{zero} & \mathtt{X} \\ \mathtt{plus}(\mathtt{suc}(\mathtt{Y}),\mathtt{X}) & \rightarrow_{plus} & \mathtt{suc}(\mathtt{plus}(\mathtt{Y},\mathtt{X})) \end{array}$$

Example 1.1.8 (Rewriting)**.** The following sequence of rewrites represent the computation of $2+1$,

$\mathtt{plus}(\mathtt{suc}(\mathtt{suc}(0)),\mathtt{suc}(0)) \rightarrow_{plus} \mathtt{suc}(\mathtt{plus}(\mathtt{suc}(0),\mathtt{suc}(0)))$
$\rightarrow_{plus} \mathtt{suc}(\mathtt{suc}(\mathtt{plus}(0,\mathtt{suc}(0)))) \rightarrow_{zero} \mathtt{suc}(\mathtt{suc}(\mathtt{suc}(0)))$.

1.2 Nominal Rewriting

Nominal rewrite sytems (NRSs) were introduced in (Fernández and Gabbay, 2007; Fernández et al., 2004) as a generalisation of TRSs providing support for binding structures using the nominal approach. Below, we recall its syntax and main concepts.

1.2.1 Nominal terms

A *nominal signature* Σ is a set of term-formers, or *function symbols*, $f, g, \ldots$, each with a fixed arity. Fix a countably infinite set $\mathscr{X}$ of *variables* ranged over by $X, Y, Z, \ldots$, and a countably infinite set $\mathscr{A}$ of *atoms* ranged over by $a, b, c, \ldots$, and assume that Σ, $\mathscr{X}$ and $\mathscr{A}$ are pairwise disjoint.

The following signatures are inspired by examples from (Fernández and Gabbay, 2007; Fernández et al., 2004) and will be applied to recurrent examples during this and future chapters.

Example 1.2.1. The nominal signature for the untyped lambda calculus has function symbols

$$\texttt{abs}:1 \quad \texttt{app}:2 \quad \texttt{subst}:2$$

where notation _:_ relates the function symbol to its fixed arity.

The signature for the differentiation operator contains the following function symbols

$$\texttt{mult}:2 \quad \texttt{diff}:1 \quad \texttt{sin}:1 \quad \texttt{cos}:1 \quad \texttt{app}:2.$$

To compute prenex normal forms in first-order logic we need the following signatures

$$\texttt{and}:2 \quad \texttt{or}:2 \quad \texttt{not}:1 \quad \texttt{forall}:1 \quad \texttt{exists}:1.$$

The following remark is implicitly applied along this and following chapters.

Remark 1.2.2 (Permutative convention notation). We follow the **permutative convention** (Gabbay and Mathijssen, 2008, Convention 2.3) for atoms throughout the paper, that is, atoms a, b, c range permutatively over $\mathscr{A}$ so that they are always distinct among them unless stated otherwise.

Definition 1.2.3 (Permutation of atoms)**.** A **permutation** π is a bijection on $\mathscr{A}$ such that the set of atoms for which $a \neq \pi(a)$ is finite. This set is called the **support** of π, written $\mathscr{S}upport(\pi)$.

A **swapping** is a pair of atoms, written $(a\ b)$, where a maps to b, b maps to a and all other c map to themselves. Then, a permutation π is represented by a list of swappings applied in the right-to-left order and generated by the grammar $\pi ::= \mathtt{Id} \mid \pi(a\ b)$, where $\mathtt{Id}$ denotes the **identity permutation** and is commonly omitted.

Write $\pi' \circ \pi$ for **composition** of permutations such that $(\pi' \circ \pi)(a) = \pi'(\pi(a))$ and π^{-1} for the **inverse** of π, that is, if $\pi = (a\ b)(b\ c)$ then $\pi(c) = a$ if and only if $c = \pi^{-1}(a)$.

Intuitively, we are only interested on those names in the representation of a permutation which play an active role in the replacement of atoms. As a result, a bijection π does not have a unique syntactic representation. For instance $(a\ b)(c\ d)(d\ c) = (a\ b)$ since $\mathscr{S}upport((a\ b)(c\ d)(d\ c)) = \mathscr{S}upport((a\ b)) = \{a,b\}$.

We are ready to introduce the grammar of nominal terms. The syntax is based on (Fernández et al., 2004; Urban et al., 2004).

Definition 1.2.4 (Syntax)**.** *Nominal terms*, or just *terms*, are generated by the grammar

$$s,t ::= a \mid \pi{\cdot}X \mid [a]s \mid fs \mid (s_1,\ldots,s_n)$$

and called, respectively, atoms, moderated variables or simply variables, abstractions, function applications (which must respect the arity of the function symbol) and tuples (if $n = 0$ or $n = 1$ we may omit the parentheses). We abbreviate $\mathtt{Id}{\cdot}X$ as X if there is no ambiguity. An abstraction $[a]t$ is intended to represent t with a bound; we say t is in the scope of $[a]$. Call occurrences of a *abstracted* if they are in the scope of an abstraction for a and *unabstracted* (or free) otherwise.

For example, $f(X,(a\ b){\cdot}X)$ is a nominal term, and so is $f([a]X,[b]b)$. The latter term has X in the scope of $[a]$ and b in the scope of $[b]$. For more examples, we refer the reader to (Fernández and Gabbay, 2007; Urban et al., 2004).

Permutation action

We now extend the action of permutations to terms. Recall we use the permutative convention from Remark 1.2.2, so atoms a,b,c are considered distinct among them.

Definition 1.2.5 (Permutation action)**.** The *action of a permutation π on a term t*, written $\pi{\cdot}t$, is defined by induction: $\mathtt{Id}{\cdot}t = t$ and $(a\ b)\pi{\cdot}t = (a\ b){\cdot}(\pi{\cdot}t)$, where a swapping acts

inductively on the structure of terms as follows:

$$(a\ b)\cdot a = b \quad (a\ b)\cdot b = a \quad (a\ b)\cdot c = c$$
$$(a\ b)\cdot(\pi\cdot X) = ((a\ b)\circ\pi)\cdot X \qquad (a\ b)\cdot[c]t = [c](a\ b)\cdot t$$
$$(a\ b)\cdot[a]t = [b](a\ b)\cdot t \qquad (a\ b)\cdot[b]t = [a](a\ b)\cdot t$$
$$(a\ b)\cdot ft = f(a\ b)\cdot t \qquad (a\ b)\cdot(t_1,\ldots,t_n) = ((a\ b)\cdot t_1,\ldots,(a\ b)\cdot t_n).$$

Example 1.2.6 (Permutation action)**.**

$$(\mathtt{a}\ \mathtt{b})\cdot\mathtt{and}(\mathtt{or}(\mathtt{b},\mathtt{c}),\mathtt{forall}([\mathtt{a}]\mathtt{P})) = \mathtt{and}(\mathtt{or}(\mathtt{a},\mathtt{c}),\mathtt{forall}([\mathtt{b}](\mathtt{a}\ \mathtt{b})\cdot\mathtt{P}))$$

Positions and subterms of nominal terms

Definition 1.2.7. The functions $V(t)$ and $A(t)$ are used to compute the sets of variables and atoms in a term t, respectively. They are inductively defined as follows:

$$V(a) = \varnothing \qquad V([a]t) = V(t) \qquad V(\pi\cdot X) = \{X\}$$
$$V(fs) = V(s) \qquad V((s_1,\ldots,s_n)) = V(s_1)\cup\ldots\cup V(s_n)$$

$$A(a) = \{a\} \qquad A([a]t) = A(t)\cup\{a\} \qquad A(\pi\cdot X) = \mathscr{S}upport(\pi)$$
$$A(fs) = A(s) \qquad A((s_1,\ldots,s_n)) = A(s_1)\cup\ldots\cup A(s_n)$$

Ground terms have no variables: $V(t) = \varnothing$.

Outermost brackets are commonly omitted when $V(\cdot)$ (resp. $A(\cdot)$) is applied to a tuple, that is, $V((s_1,\ldots,s_n))$ (resp. $A((s_1,\ldots,s_n))$) is thus denoted as $V(s_1,\ldots,s_n)$ (resp. $A(s_1,\ldots,s_n)$).

Notice that $V(t)$ is a syntactic notion, whereas $A(t)$ takes into account the semantics of permutations (represented as lists of swappings). For example, $A(f(a,\,[b](c\ d)(e\ f)(f\ e)\cdot X))$ $= \{a,b,c,d\}$.

Definition 1.2.8 (Positions and subterms of nominal terms)**.** Let s be a nominal term. The set $\mathscr{P}os(s)$ of *positions* in s is a set of strings of positive integers, inductively defined below. We also define below the subterms of s: $s|_p$ denotes the subterm of s at position p.

- if $s = a$ or $s = \pi\cdot X$, then $\mathscr{P}os(s) = \{\varepsilon\}$ and $s|_\varepsilon = s$, where ε denotes the empty string;
- if $s = [a]t$, then $\mathscr{P}os(s) = \{\varepsilon\}\cup\{1\cdot p \mid p\in\mathscr{P}os(t)\}$, $s|_\varepsilon = s$ and $s|_{1\cdot p} = t|_p$;
- if $s = ft$, then $\mathscr{P}os(s) = \{\varepsilon\}\cup\{1\cdot p \mid p\in\mathscr{P}os(t)\}$, $s|_\varepsilon = s$ and $s|_{1\cdot p} = t|_p$;

- if $s = (t_1, \ldots, t_n)$, then $\mathscr{P}os(s) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i \cdot p \mid p \in \mathscr{P}os(t_i)\}$, $s|_\varepsilon = s$ and $s|_{i \cdot p} = t_i|_p$.

The position ε is called the *root position* of the term s, and the symbol at this position is called the *root symbol* of s.

Example 1.2.9 (Sets of atoms and variables in a term)**.** The term $\mathtt{t} = \mathtt{app}([\mathtt{a}](\mathtt{b}\ \mathtt{c}) \cdot \mathtt{X}, \mathtt{Y})$ has $A(\mathtt{t}) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and and $V(\mathtt{t}) = \{\mathtt{X}, \mathtt{Y}\}$.

Substitution of variables by terms

Substitution of variables by terms is a syntactic meta-level transformation on the structure of nominal terms.

Definition 1.2.10 (Substitution)**.** *Substitutions* are generated by the grammar:

$$\sigma ::= \mathtt{Id} \mid [X \mapsto s]\sigma$$

where $\mathtt{Id}$ is commonly omitted.

Substitution lists are interpreted as a set of simultaneous mappings acting on variables without avoiding capture of atoms. We use the same notation for the identity substitution and permutation since there will be no ambiguity.

The image of a variable symbol X under v-substitution σ is depicted as $\sigma(X)$ and known as an **instantiation** of X by σ. Write **domain of** σ, $\mathscr{D}om(\sigma)$, for the set of variables such that $\sigma(X) \neq X$ for all $X \in \mathscr{X}$. Call **image of** σ, written $\mathscr{I}mg(\sigma)$, the set of terms $\{\sigma(X) \mid X \in \mathscr{D}om(\sigma)\}$. The **restriction of a substitution** σ to a set of variables V, written $\sigma|_V$, is defined as $\sigma|_V = [X \mapsto \sigma(X) \mid X \in V]$.

Write $t\sigma$ for the application of σ on t, defined as follows:

$$t\,\mathtt{Id} \triangleq t \qquad a[X \mapsto s] \triangleq a \qquad (\pi \cdot X)[X \mapsto s] \triangleq \pi \cdot s$$
$$(\pi \cdot Y)[X \mapsto s] \triangleq \pi \cdot Y \ \ (X \neq Y) \qquad ([a]t)[X \mapsto s] \triangleq [a](t[X \mapsto s])$$
$$(ft)[X \mapsto s] \triangleq ft[X \mapsto s] \qquad (t_1, \ldots, t_n)[X \mapsto s] \triangleq (t_1[X \mapsto s], \ldots, t_n[X \mapsto s]).$$

Call a term t an **instance** of a term s when t is the result of applying σ to s, $s\sigma$.

Composition of variable substitutions, written $\sigma \bullet \sigma'$, is applied inversely since the notation is postfix. Therefore $s(\sigma \bullet \sigma') = (s\sigma)\sigma'$.

In the sequel, a sequential list of simultaneous v-substitution mappings $[X_1 \mapsto s_1] \cdots [X_n \mapsto s_n]$ is represented as $[X_1 \mapsto s_1; \ldots; X_n \mapsto s_n]$.

Example 1.2.11.

$$(\mathtt{map}([\mathtt{a}]\mathtt{F},\mathtt{cons}([\mathtt{b}]\mathtt{H},\mathtt{T})))[\mathtt{F}\mapsto\mathtt{suc}(\mathtt{a});\mathtt{H}\mapsto\mathtt{g}(\mathtt{a})]=\mathtt{map}([\mathtt{a}]\mathtt{suc}(\mathtt{a}),\mathtt{cons}([\mathtt{b}]\mathtt{g}(\mathtt{a}),\mathtt{T})).$$

Observe how atom a in the image of variable F under the substitution is captured by the abstraction for a following the substitution action.

Next, we provide a definition of α-equivalence for nominal terms.

Freshness and α-equality of nominal terms

The semantics of nominal terms is defined using nominal sets. A $Perm(\mathscr{A})$-set is a set T equipped with a permutation action, such that $\mathtt{Id}\cdot x = x$ and $\pi\cdot(\pi'\cdot x) = (\pi\circ\pi')\cdot x$ for each object $x\in T$. A set S of atoms *supports* $x\in T$ if for all atoms $a,b\notin S$, $(a\ b)\cdot x = x$. A *nominal set* is a $Perm(\mathscr{A})$-set where each element has finite support. Nominal terms form a nominal set, using α-equivalence as equality (Pitts, 2013). To define the support of a term, we introduce the notion of *freshness*.

Definition 1.2.12 (Freshness)**.** A *freshness constraint* is a pair $a\#t$ of an atom and a term. A *freshness context* (ranged over by Δ,∇,Γ), is a set of constraints of the form $a\#X$. *Freshness judgements*, written $\Delta\vdash a\#t$, are derived using the rules below.

$$\frac{}{\Delta\vdash a\#b}\ (\#\mathbf{ab}) \qquad \frac{\pi^{-1}\cdot a\#X\in\Delta}{\Delta\vdash a\#\pi\cdot X}\ (\#\mathbf{X}) \qquad \frac{\Delta\vdash a\#s}{\Delta\vdash a\#fs}\ (\#\mathbf{f})$$

$$\frac{\Delta\vdash a\#s_1\ \cdots\ \Delta\vdash a\#s_n}{\Delta\vdash a\#(s_1,\ldots,s_n)}\ (\#\mathbf{tupl}) \qquad \frac{}{\Delta\vdash a\#[a]s}\ (\#[\mathbf{a}]) \qquad \frac{\Delta\vdash a\#s}{\Delta\vdash a\#[b]s}\ (\#[\mathbf{b}])$$

Example 1.2.13. Given $\nabla=\{a\#X\}$, we can derive $\nabla\vdash b\#(a\ b)\cdot X$ as follows.

$$\frac{a\#X\in\nabla}{\nabla\vdash b\#(a\ b)\cdot X}\ (\#\mathbf{X})$$

Definition 1.2.14 (Support of a term)**.** The support set of a term t, abbreviated $supp(t)$, is the complement of the set of fresh atoms in t.

When a term t is ground, $supp(t)$ coincides with the syntactic notion of unabstracted atoms in t.

In nominal languages, one is interested in terms t that have finite support, because for them there exists always a fresh atom a such that $a\#t$ (recall the set $\mathscr{A}$ of atoms is infinite).

Freshness and α-equality are fundamental concepts of formal languages with variable-binding constructs. As we have shown above, freshness is used to indicate which atoms do not occur free in terms. The concept of α-equality for nominal terms coincide with the concept of α-conversion for formalisms manipulating higher-order terms, that is, syntax trees built with higher-order functions and object-level variables that can be free or bound. The novelty of α-equality of terms employing the nominal approach is the use of the auxiliary freshness relation given above to characterise freshness of bound names in terms and also the operation of swapping pairs of atoms. Then, by means of a small logic one can inductively define an explicit operation of α-conversion which takes into account certain symmetries on terms induced by atom swapping, as we show next.

Definition 1.2.15 (α-equivalence)**.** An *α-equality constraint* is a pair $s \approx_\alpha t$ of terms. *Equivalence judgements*, written $\Delta \vdash s \approx_\alpha t$, are derived using the rules below, where $\mathtt{ds}(\pi,\pi') = \{a \in \mathscr{A} \mid \pi{\cdot}a \neq \pi'{\cdot}a\}$ (difference set).

$$\frac{}{\Delta \vdash a \approx_\alpha a}\ (\approx_\alpha \mathbf{a}) \qquad \frac{\forall a \in \mathtt{ds}(\pi,\pi') : a\#X \in \Delta}{\Delta \vdash \pi{\cdot}X \approx_\alpha \pi'{\cdot}X}\ (\approx_\alpha \mathbf{X}) \qquad \frac{\Delta \vdash s \approx_\alpha t}{\Delta \vdash fs \approx_\alpha ft}\ (\approx_\alpha \mathbf{f})$$

$$\frac{\Delta \vdash s_1 \approx_\alpha t_1 \cdots \Delta \vdash s_n \approx_\alpha t_n}{\Delta \vdash (s_1,\ldots,s_n) \approx_\alpha (t_1,\ldots,t_n)}\ (\approx_\alpha \mathbf{tupl})$$

$$\frac{\Delta \vdash s \approx_\alpha t}{\Delta \vdash [a]s \approx_\alpha [a]t}\ (\approx_\alpha [\mathbf{a}]) \qquad \frac{\Delta \vdash (b\ a){\cdot}s \approx_\alpha t \quad \Delta \vdash b\#s}{\Delta \vdash [a]s \approx_\alpha [b]t}\ (\approx_\alpha [\mathbf{b}])$$

Let P_i be a freshness or α-equality constraint (for $1 \leq i \leq n$). We write $\Delta \vdash P_1,\ldots,P_n$ when proofs of $\Delta \vdash P_i$ exist (for $1 \leq i \leq n$), using the derivation rules above.

The relation $\approx_\alpha$ is indeed an equivalence relation (Urban et al., 2004).

Example 1.2.16. Given $\nabla = \{a\#X\}$, we can derive $\nabla \vdash [a](a\ b){\cdot}X \approx_\alpha [b]X$ as follows.

$$\frac{\dfrac{ds((b\ a)(a\ b),\mathtt{Id}) = \varnothing}{\nabla \vdash (b\ a)(a\ b){\cdot}X \approx_\alpha X}\ (\approx_\alpha \mathbf{X}) \qquad \dfrac{}{\nabla \vdash b\#(a\ b){\cdot}X}\ (\#\mathbf{X})}{\nabla \vdash [a](a\ b){\cdot}X \approx_\alpha [b]X}\ (\approx_\alpha [\mathbf{b}])$$

Lemma 1.2.17 (Strengthening (Fernández and Gabbay, 2010), Lemma 2.11)**.** *Suppose $a \notin A(s,t)$. Then,*

- $\Delta, a\#X \vdash b\#s$ *implies* $\Delta \vdash b\#s$*;*
- $\Delta, a\#X \vdash s \approx_\alpha t$ *implies* $\Delta \vdash s \approx_\alpha t$*.*

Lemma 1.2.18 (Weakening (Fernández and Gabbay, 2010), Lemma 2.13)**.** *For any freshness context* Γ*,*

- $\Delta \vdash b\#s$ *implies* $\Delta, \Gamma \vdash b\#s$*;*
- $\Delta \vdash s \approx_\alpha t$ *implies* $\Delta, \Gamma \vdash s \approx_\alpha t$*.*

The following pair of results are main technical properties of nominal terms. In Property 1.2.19, we state that α-equivalent terms have the same support set. Property 1.2.20 declares that the freshness and α-equivalence relation are preserved under the action of permutations.

Property 1.2.19 ((Fernández and Gabbay, 2007), Lemma 23)**.** *For any* $a \in \mathscr{A}$*, if* $\Delta \vdash a\#s$ *and* $\Delta \vdash s \approx_\alpha t$ *then* $\Delta \vdash a\#t$*. Hence, if* $\Delta \vdash s \approx_\alpha t$ *then* s *and* t *have the same support set.*

Property 1.2.20 ((Fernández and Gabbay, 2007), Lemma 20)**.** *If* $\nabla \vdash a\#t$ *then* $\nabla \vdash \pi \cdot a\#\pi \cdot t$*. Similarly, if* $\nabla \vdash s \approx_\alpha t$ *then* $\nabla \vdash \pi \cdot s \approx_\alpha \pi \cdot t$*.*

Terms-in-context

Terms-in-context are a key concept in the definition of nominal rewriting.

Definition 1.2.21. A **term-in-context**, $\nabla \vdash s$, is a pair of a freshness context ∇ and a nominal term s.

We have shown nominal terms to be regarded as first-order terms with built-in α-conversion. We continue by providing a notion of nominal matching in order to mechanise the rewriting of terms-in-context.

1.2.2 Nominal unification and nominal matching

Nominal unification is a generalisation of first order unification using the nominal approach. Given a pair of nominal terms s and t, the unification algorithm given in (Urban et al., 2004) decides whether there exists a pair of a substitution σ and a freshness context Δ that makes s and t equivalent in the sense of relation $\approx_\alpha$ as in Definition 1.2.15. Unifying a pair of nominal terms may introduce freshness conditions when resolving equality between abstractions with distinct names as in rule $(\approx_\alpha[\mathbf{b}])$. Then, a unification algorithm solves not only unification but also freshness constraints. However, freshness constraints are not solved by instantiation but they provide soundness of the unifiers generated during the unification process.

We begin by describing the unification problem.

Definition 1.2.22 (Nominal unification and its solution)**.** A **unification problem** Pr is a finite set of constraints over a signature, each of which is either a **unification constraint** $s \stackrel{?}{\approx}_? t$ where s,t are a terms over the signature, or a freshness constraint $a\#t$ where a is an atom and t a term over the signature.

A **solution** for Pr, $\langle Pr \rangle_{sol}$, is a pair (Δ, σ) of a freshness context Δ and a substitution σ satisfying

- $\Delta \vdash a\#t\sigma$ for each $a\#t \in Pr$ and
- $\Delta \vdash s\sigma \approx_\alpha t\sigma$ for each $s \approx_\alpha t \in Pr$.
- $\Delta \vdash X(\sigma \bullet \sigma) \approx_\alpha X\sigma$ for each $X \in \mathscr{D}om(\sigma)$. We say σ is idempotent.

Example 1.2.23. The pair $(\{\mathtt{a\#Y}\}, \sigma = [\mathtt{X} \mapsto (\mathtt{a\ b})\cdot\mathtt{Y}])$ is a solution to the unification problem

$$\{[\mathtt{a}]\mathtt{X} \stackrel{?}{\approx}_? [\mathtt{b}]\mathtt{Y}\}$$

since $\mathtt{a\#Y} \vdash ([\mathtt{a}]\mathtt{X})[\mathtt{X} \mapsto (\mathtt{a\ b})\cdot\mathtt{Y}] \stackrel{?}{\approx}_? ([\mathtt{b}]\mathtt{Y})[\mathtt{X} \mapsto (\mathtt{a\ b})\cdot\mathtt{Y}]$ holds.

There can be more than one solution to the same unification problem. Write $\mathscr{U}(Pr)$ for the set of all solutions to some unification problem Pr. Solutions can be compared using the following ordering relation (Fernández and Gabbay, 2007).

Definition 1.2.24 (Ordering relation)**.** Let Δ_1, Δ_2 be a pair of freshness contexts and σ_1, σ_2 a pair of substitutions. Then, $(\Delta_1, \sigma_1) \leq (\Delta_2, \sigma_2)$ when there is some θ such that for all X,

$$\Delta_2 \vdash X(\sigma_1 \bullet \theta) \approx_\alpha X\sigma_2 \quad \text{and} \quad \Delta_2 \vdash \Delta_1\theta.$$

Nominal unification enjoys uniqueness of **principal solutions** (Urban et al., 2004). $(\Delta, \sigma) \in \mathscr{U}(Pr)$ is a principal solution for Pr if any other solution $(\nabla, \theta) \in \mathscr{U}(Pr)$ is an instance of (Δ, σ), that is, there exists a substitution σ' satisfying $\nabla \vdash \Delta\sigma'$ (here $\Delta\sigma' = \{a\#\sigma'(X) \mid a\#X \in \Delta\}$) and also $\nabla \vdash X(\sigma \bullet \sigma') \approx_\alpha X\theta$ for $X \in \mathscr{X}$.

The naive implementation given in (Urban et al., 2004), representing terms as trees, is exponential. Cheney proved that a more general form, *equivariant unification*, is NP-complete (Cheney, 2010). However, nominal unification has proven to be quadratic in time and space by two different implementations (Calvès, 2010; Calvès and Fernández, 2009) and (Levy and Villaret, 2010) inspired by the Paterson-Wegman unification algorithm. In (Calvès, 2013), Calvés demonstrates that both implementations can be unified. Nominal unification is used in α-Prolog (Cheney and Urban, 2004).

The section continues by providing a description of the problem of matching using the nominal approach.

Nominal matching

Nominal matching is a simpler version of nominal unification. A **matching constraint** $s \stackrel{?}{\approx} t$ can be seen as a particular kind of unification constraint where term t is *ground*, that is, it has no variables. More generally, if there are variables occurring in t, then the variables in t cannot be instantiated. The left-hand side of a matching constraint $s \stackrel{?}{\approx} t$ is called a *pattern*. Matching has applications in functional programming and rewriting, among others.

The following definition will be of use when describing nominal matching and nominal rewriting (see Section 1.2).

Definition 1.2.25. We extend the notions given in Definition 1.2.7 for both variables, $V(t)$, and atoms, $A(t)$, to include terms-in-context, freshness contexts and substitutions. Particularly, for freshness contexts $A(\Delta) = \{a \mid a\#X \in \Delta \text{ for some } X\}$ and for substitutions, $A(\sigma) = \{A(X\sigma) \mid X \in dom(\sigma)\}$. Then, for terms-in-context we have, $A(\Delta \vdash t) = A(\Delta) \cup A(t)$.

Similarly for function $V(\cdot)$. We omit the definitions.

Definition 1.2.26 (Matching problem-in-context)**.** A **matching problem-in-context** consists of a pair of terms-in-context $\nabla \vdash l$ and $\Delta \vdash t$ where $V(\nabla \vdash l) \cap V(\Delta \vdash t) = \varnothing$. A matching problem-in-context is written

$$(\nabla \vdash l) \stackrel{?}{\approx} (\Delta \vdash t)$$

The **solution** to this matching problem-in-context, if exists, is a substitution σ such that:

- $\langle \nabla \cup \{l \stackrel{?}{\approx}_? s\} \rangle_{sol} = (\Gamma, \sigma)$ where
- $\Delta \vdash \Gamma$ and also
- $X\sigma = X$ for $X \in V(\Delta \vdash t)$.

We say that σ **solves** the matching problem.

Informally, a nominal matching problem-in-context is a unification problem containing one unification constraint and (possibly none) freshness constraints where one adds the restriction that the variables in the LHS of the unification constraint are disjoint from the variables in the RHS and that only variables in the LHS may be instantiated. Then, a solution to a matching problem-in-context is a solution to a unification problem where the restrictions aforementioned are satisfied and, additionally, where the set of freshness contexts from the solution pair is derivable from the set of freshness contexts in the matched term-in-context.

Example 1.2.27. The matching problem-in-context

$$(a\#X \vdash [a]X) \stackrel{?}{\approx} (\vdash [a]a)$$

has no solution.

Nominal matching can be solved in linear time (Calvès, 2010). Nominal rewriting is implemented with the nominal matching algorithm (Fernández and Gabbay, 2007; Fernández et al., 2004).

Next, we recall the particularities of rewriting in the nominal framework.

1.2.3 Nominal Rewrite Systems

Definition 1.2.28 (Nominal rewrite system)**.** A **nominal rewrite rule** R, or **rewrite rule** for short, is a tuple consisting of a freshness context ∇ and terms l and r such that $V(r) \cup V(\nabla) \subseteq V(l)$. A rewrite rule is written $\nabla \vdash l \to r$.

A *nominal rewrite system* $\mathscr{R}$ is a pair (Σ, Rw) of a signature Σ and a possibly infinite set of nominal rewrite rules Rw over Σ.

We may omit Σ, identifying R with Rw when the signature is clear from the context.

Example 1.2.29. The following rules are used to compute prenex normal forms in first-order logic using the signature from Example 1.2.1.

$$
\begin{array}{lll}
a\#P & \vdash & \mathsf{and}(P, \mathsf{forall}([a]Q)) \to \mathsf{forall}([a]\mathsf{and}(P,Q)) \\
a\#P & \vdash & \mathsf{and}(\mathsf{forall}([a]Q), P) \to \mathsf{forall}([a]\mathsf{and}(Q,P)) \\
a\#P & \vdash & \mathsf{or}(P, \mathsf{forall}([a]Q)) \to \mathsf{forall}([a]\mathsf{or}(P,Q)) \\
a\#P & \vdash & \mathsf{or}(\mathsf{forall}([a]Q), P) \to \mathsf{forall}([a]\mathsf{or}(Q,P)) \\
a\#P & \vdash & \mathsf{and}(P, \mathsf{exists}([a]Q)) \to \mathsf{exists}([a]\mathsf{and}(P,Q)) \\
a\#P & \vdash & \mathsf{and}(\mathsf{exists}([a]Q), P) \to \mathsf{exists}([a]\mathsf{and}(Q,P)) \\
a\#P & \vdash & \mathsf{or}(P, \mathsf{exists}([a]Q) \to \mathsf{exists}([a]\mathsf{or}(P,Q)) \\
a\#P & \vdash & \mathsf{or}(\mathsf{exists}([a]Q), P) \to \mathsf{exists}[a]\mathsf{or}(Q,P) \\
 & \vdash & \mathsf{not}(\mathsf{exists}([a]Q)) \to \mathsf{forall}([a]\mathsf{not}(Q)) \\
 & \vdash & \mathsf{not}(\mathsf{forall}([a]Q)) \to \mathsf{exists}([a]\mathsf{not}(Q)).
\end{array}
$$

In (Fernández and Gabbay, 2007; Fernández et al., 2004) nominal rewrite systems have the additional property of being **equivariant**, that is, if a rewrite rule R exists in the set of rewrite rules Rw then also $R^\pi \in Rw$ where R^π is R with π applied to all atoms. This is known as a *meta-permutation action*. Then, given a set of rewrite rules Rw, a nominal rewrite system contains all the permuted variants of each rule in Rw, what is known as the **equivariant closure** of Rw. We follow the approach from (Fernández and Gabbay, 2010) where equivariance is explicitly introduced in the *nominal rewrite relation* by means of a meta-permutation π. However, the property of equivariance have the same effect as the

introduction of the meta-level permutation π in the nominal rewrite relation (Fernández and Gabbay, 2010).

Prior the definition of nominal rewriting we formalise the action of meta-permutations.

Definition 1.2.30 (Meta-permutation action)**.** Let t be an nominal term and π a meta-permutation. Then, the **meta-application** of π on t, denoted as t^π, is inductively defined as follows:

$$a^\pi \triangleq \pi(a) \quad ([a]t)^\pi \triangleq [a^\pi]t^\pi \quad (s_1,\ldots,s_n)^\pi \triangleq (s_1^\pi,\ldots,s_n^\pi)$$

$$(ft)^\pi \triangleq ft^\pi \quad (\pi_1{\cdot}X)^\pi \triangleq \pi_1^\pi{\cdot}X \quad \text{where } \pi_1^\pi \triangleq \begin{cases} (a^\pi\ b^\pi)\circ\tau^\pi, & \text{if } \pi_1 = (a\ b)\circ\tau \\ \mathtt{Id}, & \text{if } \pi_1 = \mathtt{Id} \end{cases}$$

Nominal rewriting (Fernández and Gabbay, 2007) operates on terms-in-contexts $\Delta \vdash s$ or just s if $\Delta = \varnothing$.

Definition 1.2.31 (Nominal rewriting)**.** A rewrite system $\mathscr{R}$ induces a **one-step rewrite relation** $\Delta \vdash s \xrightarrow{R} t$ on terms s,t as follows: $\exists(\nabla \vdash l \to r) \in \mathscr{R}, p \in \mathscr{P}os(s)$, meta-permutation π and substitution θ such that: (as usual, we assume $V(R) \cap (V(\Delta) \cup V(s)) = \varnothing$)

$$\frac{\Delta \vdash (\nabla^\pi\theta, \quad s|_p \approx_\alpha l^\pi\theta, \quad s[r^\pi\theta]_p \approx_\alpha t)}{\Delta \vdash s \xrightarrow{R} t}\ (\to\textbf{Rew})$$

Since Δ does not change during rewriting, a rewriting derivation is written $\Delta \vdash s_1 \to_R s_2 \to_R \ldots \to_R s_n$, abbreviated as $\Delta \vdash s_1 \to^* s_n$.

When rules are closed, nominal rewriting can be efficiently implemented using nominal matching (then, there is no need to consider equivariance). We define closed rewriting below, after defining closed terms.

Closed terms are, roughly speaking, terms without unabstracted atoms, such that variables behave uniformly with respect to their support. We give a definition below.

Definition 1.2.32 (Closedness)**.** A term-in-context $\Delta \vdash t$ is **closed** if it satisfies the following conditions:

1. if $t|_p = a$ then $t|_p$ is in the scope of an abstraction for a;
2. if $\pi \cdot X$ occurs in the scope of an abstraction of $\pi \cdot a$ then any occurrence of $\pi' \cdot X$ occurs in the scope of an abstraction of $\pi' \cdot a$ or $a\#X \in \Delta$;
3. for any pair $\pi_1{\cdot}X, \pi_2{\cdot}X$ occurring in t, and $a \in ds(\pi_1,\pi_2)$, if a is not abstracted in one of the occurrences then $a\#X \in \Delta$.

A rewrite rule $\nabla \vdash l \to r$ is **closed** if $\nabla \vdash (l,r)$ is a closed term.

The first condition in the definition specifies that no atom occurs unabstracted in a closed term. The second condition states that if an atom a in an instance of a variable $\pi{\cdot}X$ is captured (i.e. $\pi{\cdot}X$ is under an abstraction for $\pi{\cdot}a$) then it is captured in all occurrences of X, otherwise it is fresh for X. The third condition says that if two occurrences of X have different suspended permutations, then any atom in the difference set that could occur in an instance of X is captured.

For example, $[a]f(X,a)$ is closed, but $f(X,a)$ and $f(X,[a]X)$ are not, however $a\#X \vdash f(X,[a]X)$ is closed. All the rewrite rules in Example 1.2.29 are closed.

Definition 1.2.33. (Freshened variants) A **freshened variant** $t^\prime$ is a nominal term with the same structure as term t except that atoms and variables have been replaced by *fresh* atoms and variables, namely, atoms not in $A(t)$ and variables not in $V(t)$, and possibly also for other atoms and variables occurring in a term-in-context $\Delta \vdash s$ which we will always specify when it is not obvious. We omit an inductive definition.

Similarly, $\Delta^\prime$ denotes a freshened variant of a freshness context Δ, that is, if $a\#X \in \Delta$ then $a^\prime\#X^\prime \in \Delta^\prime$ where $a^\prime$ and $X^\prime$ are freshened variants of atoms and variables occurring in Δ.

We may extend this to other syntax, like rewrite rules. Then, $\nabla^\prime \vdash l^\prime \to r^\prime$ is a freshened variant of the rewrite rule $\nabla \vdash l \to r$ (and perhaps a given term-in-context). Note that $V(\nabla^\prime \vdash l^\prime \to r^\prime) \cap V(\nabla \vdash l \to r) = \varnothing$.

In practice, the creation of newly-freshened variants is often accomplished by the generation of new atoms and variables with respect to the entire system.

Closedness can be easily checked using the nominal matching algorithm (Calvès and Fernández, 2009) as follows. First, given a term in context $\nabla \vdash t$, or more generally, a pair $P = \nabla \vdash (l,r)$ (this could be a rule $R = \nabla \vdash l \to r$), let us write $P^\prime = \nabla^\prime \vdash (l^\prime, r^\prime)$ to denote a *freshened variant* of P, i.e., a version where the atoms and variables have been replaced by 'fresh' ones. We shall always explicitly say what $P^\prime$ is freshened for when this is not obvious. For example, a freshened version of $(a\#X \vdash f(X) \to X)$ with respect to itself and to $\vdash a'$ is $(a''\#X' \vdash f(X') \to X')$. We will write $A(P')\#V(P)$ to mean that all atoms in P' are fresh for each of the variables occurring in P. Let $\nabla^\prime \vdash t^\prime$ be a freshened version of $\nabla \vdash t$. Then $\nabla \vdash t$ is closed if there exists a substitution σ such that $\nabla, A(\nabla^\prime \vdash t^\prime)\#V(\nabla \vdash t) \vdash \nabla^\prime\sigma$ and $\nabla, A(\nabla^\prime \vdash t^\prime)\#V(\nabla \vdash t) \vdash t^\prime\sigma \approx_\alpha t$. A similar check can be done for nominal rewrite rules.

Closed rewriting follows a standard first-order rewriting notion where variables in a rewrite rule are assumed distinct from those of the term to be rewritten. Recall that higher-order rewrite systems based on λ-calculus (e.g. CRSs (Klop et al., 1993)) also follow this property, known as Barendregt's variable convention. What makes closed rewriting distinctive is that a permuted renaming is also applied to atoms. As a result, there is no interaction between atoms in the term-in-context and those explicitly named in a rule, that

is, we have chosen arbitrary atoms to denote that names in rules are sensitive to distinction among them but have no particular semantics (as we explained after Example 6.1.2). Then, we can assume that meta-permutations are trivial and find matching solutions for closed rewrites by application of the matching problem-in-context defined in Definition 1.2.26.

Definition 1.2.34 (Closed rewriting)**.** Let R^{ν} be a freshened version of the rule R with respect to Δ, s, t (i.e., a version where the atoms and variables in R have been replaced by fresh ones; as shown in (Fernández and Gabbay, 2007), it does not matter which particular freshened R^{ν} we choose). We write $\Delta \vdash s \to^{c}_{R} t$ if $\Delta, \Delta^{\phi} \vdash s \to_{R^{\nu}} t$, where $\Delta^{\phi} = A(R^{\nu}) \# V(\Delta, s)$, and call this a **closed rewriting step**. The subindex R may be omitted if it is clear from the context.

Closed NRSs inherit properties of first-order rewriting systems such as the Critical Pair Lemma (Fernández and Gabbay, 2007).

Example 1.2.35. We show a closed rewriting step for the term $\vdash \mathsf{and}(X, \mathsf{forall}([b]\mathsf{f}(b)))$ using the first rule in Example 1.2.29:

$$\vdash \mathsf{and}(X, \mathsf{forall}([b]\mathsf{f}(b))) \to^{c} \mathsf{forall}([a']\mathsf{and}(X, \mathsf{f}(a')))$$

To generate it, we first obtain a freshened variant of the rule with respect to itself and the given term: $a' \# P' \vdash \mathsf{and}(P', \mathsf{forall}([a']Q')) \to \mathsf{forall}([a']\mathsf{and}(P', Q'))$. Notice that there is a rewrite step

$$a' \# X \vdash \mathsf{and}(X, \mathsf{forall}([b]\mathsf{f}(b))) \to \mathsf{forall}([a']\mathsf{and}(X, \mathsf{f}(a'))).$$

using the matching substitution $\theta = [P' \mapsto X][Q' \mapsto f(a')]$, since $a' \# P'\theta$ holds.

1.3 Combinatory Reduction Systems

A combinatory reduction system (CRS) (Klop, 1980; Klop et al., 1993) is a pair consisting of an alphabet $\mathbb{A}$ and a set of rewrite rules.

The CRS **alphabet** $\mathbb{A}$ consists of:

1. a countably infinite set $\mathscr{V}$ of variables ranged over by $a, b, c, \ldots$;
2. a countably infinite set $\mathscr{MV}$ of meta-variables with fixed arities, written as Z_i^n where n is the arity of Z_i^n (when there is no ambiguity, n is omitted);
3. an abstraction operator $[\cdot]\cdot$;
4. function symbols $f, g, \ldots$ with fixed arities; and

5. improper symbols '(', ')'and ',',

Definition 1.3.1 (Syntax)**.** CRS **meta-terms** are generated by the grammar

$$s,t ::= a \mid Z_i^n t \mid [a]t \mid ft \mid (t_1,\ldots,t_n) \quad (n \geq 0)$$

Only variables can be abstracted; in a *function application* ft (resp. *meta-application* $Z_i^n t$), t is a n-tuple respecting the arity of the function symbol f (resp. meta-variable Z_i^n); when the arity is 0, we omit the brackets in applications and meta-applications (so Z_i^0 is a meta-term).

Definition 1.3.2. Write $MV(t)$ and $Var(t)$ for the set of meta-variables and variables occurring in a meta-term t, respectively (the same notation is used for rules, etc.). They are inductively defined as follows:

$$\begin{array}{lll} MV(a) = \varnothing & MV(Z_i^n t) = MV(t) \cup \{Z_i^n\} & MV(ft) = MV(t) \\ MV([a]t) = MV(t) & MV((t_1,\ldots,t_n)) = MV(t_1) \cup \cdots \cup MV(t_n) & \end{array}$$

$$\begin{array}{lll} Var(a) = a & Var(Z_i^n t) = Var(t) & Var(ft) = Var(t) \\ Var([a]t) = Var(t) \cup \{a\} & Var((t_1,\ldots,t_n)) = Var(t_1) \cup \cdots \cup Var(t_n) & \end{array}$$

In CRSs a distinction is made between *meta-terms* and *terms*. Meta-terms are the expressions built from the symbols in the alphabet, in the usual way (see Definition 1.3.1). Variables that occur in the scope of the abstraction operator are *bound*, and *free* otherwise. Meta-terms are defined modulo renaming of bound variables, that is, a meta-term represents an α-equivalence class. *Terms* are meta-terms that do not contain meta-variables, and are also defined modulo α-equivalence. A (meta-)term is closed if every variable occurrence is bound.

Definition 1.3.3. Let s be a CRS meta-term. The set $\mathscr{P}os(s)$ of *positions* in s is a set of strings of positive integers, which is inductively defined as follows:

- if $s = a$, then $\mathscr{P}os(s) = \{\varepsilon\}$, where ε denotes the empty string;
- if $s = Z_i^n t$ then $\mathscr{P}os(s) = \{\varepsilon\} \cup \{1 \cdot p \mid p \in \mathscr{P}os(t)\}$;
- if $s = [a]t$, then $\mathscr{P}os(s) = \{\varepsilon\} \cup \{1 \cdot p \mid p \in \mathscr{P}os(t)\}$;
- if $s = ft$, then $\mathscr{P}os(s) = \{\varepsilon\} \cup \{1 \cdot p \mid p \in \mathscr{P}os(t)\}$;
- if $s = (t_1,\ldots,t_n)$, then $\mathscr{P}os(s) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i \cdot p \mid p \in \mathscr{P}os(t_i)\}$.

The position ε is called the *root position* of the term s, and the symbol at this position is called the *root symbol* of s.

Definition 1.3.4 (CRS rewrite rules)**.** A CRS **rewrite rule** is a pair of meta-terms, written $l \Rightarrow r$, where l, r are closed, l has the form $f(s_1, \ldots, s_n)$ where $n \geq 0$ (when $n = 0$ we omit the parentheses), $MV(r) \subseteq MV(l)$, and $MV(l)$ occur only in the form $Z_i^n(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are pairwise distinct bound variables.

Example 1.3.5. The β-reduction rule for the λ-calculus is written:

$$\mathsf{app}(\mathsf{lam}([a]Z(a)), Z') \Rightarrow Z(Z')$$

where Z is a unary meta-variable and Z' is 0-ary.

The reduction relation is defined on terms. To extract from rules the actual rewrite relation, each meta-variable is replaced by a special kind of λ-term, and in the obtained term all β-redexes and the residuals of these β-redexes are reduced (i.e., a complete development is performed). Formally, the rewrite relation is defined using *substitutes* and *valuations*.

Definition 1.3.6 (Substitute)**.** An n-ary **substitute** is an expression of the form $\underline{\lambda}(a_1 \ldots a_n).s$, where s is a term and $a_1, \ldots, a_n$ are different variables bound in $\underline{\lambda}(a_1 \ldots a_n).s$. We use a meta-lambda $\underline{\lambda}$ to emphasise that this is part of the meta-language.

An n-ary substitute $\underline{\lambda}(a_1, \ldots, a_n).s$ may be applied to a n-tuple $(t_1, \ldots, t_n)$ of terms, resulting in the following simultaneous substitution:

$$(\underline{\lambda}(a_1, \ldots, a_n).s)(t_1, \ldots, t_n) = s\{a_1 \mapsto t_1, \ldots, a_n \mapsto t_n\}$$

where we denote by $s\{a \mapsto t\}$ *the capture-avoiding substitution of variable a by term t in the CRS term s.*

Definition 1.3.7 (Valuation)**.** A **valuation** σ assigns an n-ary substitute to each n-ary meta-variable:

$$\sigma(Z_i^n) = \underline{\lambda}(a_1, \ldots, a_n).s.$$

It is extended to a mapping from meta-terms to terms as follows:

1. First, replace all meta-variables in the term for their images in σ as shown below.

$$\sigma(a) = a \text{ for } a \in \mathcal{V} \qquad \sigma([a]t) = [a]\sigma(t) \qquad \sigma(ft) = f\sigma(t)$$
$$\sigma((t_1, \ldots, t_n)) = (\sigma(t_1), \ldots, \sigma(t_n)) \qquad \sigma(Z_i^n t) = \sigma(Z_i^n)\sigma(t).$$

2. Next, perform the *developments* of the β-redexes created.

Valuations must satisfy some *safety conditions*. Before stating the conditions, we recall a standard naming convention used in CRSs, originally stated by Barendregt for the λ-calculus (Barendregt, 1984).

Remark 1.3.8 (Barendregt's variable convention). CRSs adopt the following naming conventions:

- all bound variables are chosen to be different among them, that is, each binder uses a distinct variable name;
- bound variables are also chosen to be different from free variables.

In CRSs, rewriting is performed under the following conditions.

Definition 1.3.9 (Safety conditions)**.** The CRS rule $l \Rightarrow r$ *is safe for* the valuation σ if free variables occurring in substitute $\sigma(Z)$ are different from the bound variables in both l, r for all $Z \in dom(\sigma)$ and also, we say σ is safe with respect to itself when there are no two substitutes $\sigma(Z), \sigma(Z')$ where a free variable in $\sigma(Z)$ occurs bound in $\sigma(Z')$ or vice versa.

In the rest of the paper we adopt, without loss of generality, Barendregt's convention for CRSs and assume that all valuations are safe with respect to themselves and the reduction rules to which they are applied.

A **context** is a term with an occurrence of a special symbol $[\,]$ called hole. A rewrite step is now defined in the usual way.

Definition 1.3.10 (Rewrite step)**.** Let σ be a valuation and $C[\,]$ a context. If $l \Rightarrow r$ is a rewrite rule, then $C[\sigma(l)] \Rightarrow C[\sigma(r)]$ is a **rewrite (or reduction) step**.

Example 1.3.11. The following is a rewrite step using the β-rule given in Example 1.3.5:

$$\mathsf{app}(\mathsf{lam}([a]f(a,a)),t) \Rightarrow_\beta f(t,t).$$

To generate the reduction, a valuation σ that maps Z to $\underline{\lambda}(b).f(b,b)$ and Z' to the term t is applied to the rule. Then, $\sigma(\mathsf{app}(\mathsf{lam}([a]Z(a)),Z'))$ is the term $\mathsf{app}(\mathsf{lam}([a]f(a,a)),t)$ obtained by first replacing Z and Z' as indicated by σ and then reducing the β-redex $(\underline{\lambda}(b).f(b,b))(a)$. Also, $\sigma(Z(Z'))$ is the term $f(t,t)$ obtained by first replacing Z and Z', resulting in $(\underline{\lambda}(b).f(b,b))(t)$, and then β-reducing to $f(b,b)\{b \mapsto t\}$.

Summary of Our Published Work

Below, there is a summary of the work published both in (Domínguez and Fernández, 2014) and (Domínguez and Fernández, 2015) which is also part of this thesis. This summary illustrates the relationship between CRSs and NRSs by providing a pair of reduction-preserving translations from closed NRS rules and ground terms to CRS rules and terms and from CRSs to NRS. The encoding of CRS in the latter is extended with a set of explicit substitution rules to simulate higher-order substitution in the nominal rewriting framework and it is the motivation for upcoming chapters where NRS is extended with implicit substitution capabilities to simulate *and reflect* the CRS rewrite relation, that is, there is a one-to-one correspondence between a CRS one-rewrite step and a NRS one. The translation between NRSs to CRSs has been implemented in (Domínguez, 2014).

The chapter is divided into two main sections, the first section shows the translation and properties for the encoding of NRS into CRS whereas the second section does the opposite, that is, from CRS to NRS.

1.4 From NRS to CRS systems

The section begins by defining the translation of nominal terms into CRS (meta-)terms.

First, we define an auxiliary function Λ to compute, for each variable occurring in a nominal term, the set of atoms that may be captured when instantiated.

Definition 1.4.1 (Mapping Λ_t)**.** For each nominal term t, we define $\Lambda_t : X \to \mathscr{P}(\mathscr{A})$ such that $\Lambda_t(X) = \{a_1, ..., a_n\}$ if $X \in V(t)$ has k occurrences in t, A_i is the set of atoms abstracted above the ith occurrence of X, and $\{a_1, ..., a_n\} = A_1 \cup \ldots \cup A_k$. In other words, $\Lambda_t(X)$ is the set of all the atoms abstracted above occurrences of X in t. We omit the inductive definition.

For example, if $t = ([a]X, [b]X, [c]Y)$ then $\Lambda_t(X) = \{a, b\}$ and $\Lambda_t(Y) = \{c\}$.

Definition 1.4.2 (Term Translation)**.** Let $\Delta \vdash t$ be a nominal term-in-context and Λ_t as in Definition 1.4.1. Then $\mathscr{T}(\Delta, t) = [\![t]\!]^{\Delta}_{\Lambda_t}$, where $[\![\cdot]\!]^{\Delta}_{\Lambda_t}$ is an auxiliary function defined by induction over the structure of nominal terms as follows:

(atom)	$[\![a]\!]^{\Delta}_{\Lambda_t}$	=	a,
(var)	$[\![\pi{\cdot}X]\!]^{\Delta}_{\Lambda_t}$	=	$X(\overline{xs})$ where
			$\overline{xs} \triangleq \pi \cdot xs$ (we omit $(\overline{xs})$ if empty)
			$xs \triangleq \texttt{toAscList}([\pi^{-1}{\cdot}\Lambda_t(X)] - \{a \,\vert\, a\#X \in \Delta\})$
(abs)	$[\![[a]s]\!]^{\Delta}_{\Lambda_t}$	=	$[a][\![s]\!]^{\Delta}_{\Lambda_t}$,
(fun)	$[\![fs]\!]^{\Delta}_{\Lambda_t}$	=	$f[\![s]\!]^{\Delta}_{\Lambda_t}$,
(tuple)	$[\![(s_1,\ldots,s_n)]\!]^{\Delta}_{\Lambda_t}$	=	$([\![s_1]\!]^{\Delta}_{\Lambda_t},\ldots,[\![s_n]\!]^{\Delta}_{\Lambda_t})$.

where `toAscList` is a function that builds a sorted list[1] from a set of atoms. When there is no ambiguity, we refer to the translation of a term t as $\hat{t}$.

NRS rules are more general than CRS rules in that free atoms may occur in rules.Therefore we impose some conditions on NRS rules to obtain a class of rules that can be translated to CRS rules.

Definition 1.4.3 (Standard Nominal Rule)**.** A nominal rule is called *standard* when it is closed and the left-hand side has the form fs.

Definition 1.4.4 (Rule Translation Function)**.** Let $R = \nabla \vdash l \to r$ be a standard nominal rule. The translation of R is $\mathscr{T}^{\mathscr{R}}(\nabla,l,r) = \mathscr{T}(\nabla,l) \Rightarrow \mathscr{T}(\nabla,r)$, where $\mathscr{T}(\Delta,t)$ is given in Definition 1.4.2.

Definition 1.4.5 (Substitution Translation)**.** Let $\Delta \vdash t$ be a closed nominal term-in-context, Λ_t as in Definition 1.4.1, and σ a nominal substitution satisfying Δ, such that $\sigma = [X_i \mapsto t_i]$, $1 \leq i \leq n$ where $dom(\sigma) \subseteq V(t)$ and $t\sigma$ is ground.

Then $\mathscr{T}^{\int}(\Delta,t,\sigma) = [X_i \mapsto \underline{\lambda}(\overline{xs_i}).s_i]$ is defined as follows:

- $\overline{xs_i} \triangleq \pi_i \cdot xs_i$ and,
- $xs_i \triangleq \texttt{toAscList}([\pi_i^{-1}{\cdot}\Lambda_t(X_i)] - \{a \,\vert\, a\#X_i \in \Delta\})$,
- $s_i \triangleq \mathscr{T}(\Delta, \pi_i \cdot t_i)$ where π_i is the permutation suspended in the leftmost occurrence of X_i in t.

Lemma 1.4.6 justifies the use of the leftmost occurrence of $\pi \cdot X$ in t. Intuitively, each substitute generated by application of the translation function to distinct occurrences of a moderated variable is indeed α-equivalent. Hence the leftmost occurrence is used as a representative.

We denote by $(\hat{t},\hat{\sigma})$ the result of $(\mathscr{T}(\Delta,t),\mathscr{T}^{\int}(\Delta,t,\sigma))$.

[1] List of atoms in ascending lexical order.

Lemma 1.4.6 (α-equivalence of Substitutes)**.** *Let $\Delta \vdash t$ be a closed nominal term-in-context, Λ_t as defined in Definition 1.4.1, and σ a nominal substitution satisfying Δ such that $dom(\sigma) \subseteq V(t)$ and $t\sigma$ is ground. Let $\pi_i \cdot X$, $\pi_j \cdot X$ be two occurrences of the same variable in t, and let $[X \mapsto \underline{\lambda}(\overline{xs}_i).s_i]$ and $[X \mapsto \underline{\lambda}(\overline{xs}_j).s_j]$ be translations according to Definition 1.4.5 but using π_i and π_j respectively. Then, $[X \mapsto \underline{\lambda}(\overline{xs}_i).s_i] \approx_\alpha [X \mapsto \underline{\lambda}(\overline{xs}_j).s_j]$.*

Theorem 1.4.7 (Preservation of reduction)**.** *Let $R = \nabla \vdash l \to r$ be a standard nominal rule. Let t be a ground nominal term and $\hat{t} = \mathscr{T}(\varnothing, t)$. If $t \to_R u$ then $\hat{t} \Rightarrow_{R'} \hat{u}$ using $R' = \mathscr{T}^{\mathscr{R}}(\nabla, l, r)$, and $\hat{u} = \mathscr{T}(\varnothing, u)$.*

Corollary 1.4.8 (Termination)**.** *Termination of the translated CRS implies termination of the NRS.*

1.5 From CRS to NRS systems

We begin by defining a pair of auxiliary functions.

Function Φ provides the leftmost meta-application for each meta-variable occurring in the left-hand side of a CRS rule l. More precisely, $\Phi_l(Z_i^n) = [a_1, \ldots, a_n]$ if $Z_i^n(a_1, \ldots, a_n)$ is the leftmost occurrence of Z_i^n in l. We use it in the translation to ensure the preservation of closedness and the rewriting relation (see Theorem 1.5.13).

Definition 1.5.1. Given a closed CRS meta-term t, the partial mapping Φ_t from meta-variables to lists of variables is defined such that

$$\Phi_t(Z_i^n) = [a_1, \ldots, a_n]$$

if the leftmost occurrence of the meta-variable Z_i^n in t has the form $Z_i^n(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n$ are pairwise distinct bound variables. We denote by $\Phi_t(Z_i^n)_k$ the kth element in the list $\Phi_t(Z_i^n)$.

The second auxiliary function, Ψ, is used to convert each meta-application of form $Z_i^n(a_1, \ldots, a_n)$ occurring in a left-hand side l of a CRS rule into a list of swappings suspended on a NRS variable $\pi_i \cdot Z_i$ which, when instantiated, simulates the β-reduction of a valuation σ applied to each LHS occurrence of Z_i^n. To accomplish this, Ψ is parameterised by $\Phi_l(Z_i^n)$ and applied locally to each argument list $(b_1, \ldots, b_n)$ in a meta-application of Z_i^n which is not the leftmost one, in order to preserve closedness across the NRS translation.

Definition 1.5.2. Let $s = [a_1, \ldots, a_n]$ and $t = [b_1, \ldots, b_n]$ be any two pairs of lists of length n over the set $\mathscr{V}$ of variables, and $f : [a_1, \ldots, a_n] \mapsto [b_1, \ldots, b_n]$, $f^{-1} : [b_1, \ldots, b_n] \mapsto [a_1, \ldots, a_n]$

a pair of mappings such that $b_k = f(a_k)$ and $a_k = f^{-1}(b_k)$, for $1 \leq k \leq n$. Then, $\Psi(s,t)$ returns a list π of swappings over the set of atoms $\mathscr{A}$, recursively defined as follows:

$$\begin{array}{lcl} \Psi(nil,nil) & = & \texttt{Id} \\ \Psi([a_1,\ldots,a_n],[b_1,\ldots,b_n]) & = & (a_m\ b_k)(a_m\ b_j)\cdots(a_m\ b_1)(a_m\ a_1)(a_m\ a_i)\cdots(a_m\ a_l) \\ & & \circ\ \Psi(s_1,t_1) \quad \text{where } 1 \leq i,j,k,l,m \leq n \quad \text{and} \end{array}$$

- $(a_m\ b_k)(a_m\ b_j)\cdots(a_m\ b_1)(a_m\ a_1)(a_m\ a_i)\cdots(a_m\ a_l)$ is the 2-cycle decomposition of the permutation in cycle form $C = (a_m, a_l, \ldots, a_i, a_1, b_1, \ldots, b_j, b_k)$;
- C is constructed by successive applications of functions f and f^{-1} over a_1 (as many times as possible) as follows:

$$a_m \overset{f^{-1}(a_l)}{\longleftarrow\!\shortmid} a_l \cdots \overset{f^{-1}(a_i)}{\longleftarrow\!\shortmid} a_i \overset{f^{-1}(a_1)}{\longleftarrow\!\shortmid} a_1 \overset{f(a_1)}{\longmapsto} b_1 \overset{f(b_1)}{\longmapsto} \cdots b_j \overset{f(b_j)}{\longmapsto} b_k$$

  where $f^{-1}(a_1)$ and $f(b_j)$ are only applicable when $b_k \neq a_1$. Otherwise, if $b_k = a_1$ then the cycle form would be $(a_1, b_1, \ldots, b_j)$, generating then a list of swappings $(b_j\ a_1)\cdots(b_1\ a_1)$.
- $s_1 = s \setminus C$,
- $t_1 = t \setminus C$.

To translate a CRS rule $l \Rightarrow r$, two different functions, called *Left* and *Right*, are applied to l and r respectively, both parameterised by Φ_l. We define them separately.

Definition 1.5.3 (Left Translation)**.** Let s be a closed CRS meta-term where all the meta-applications have the form $Z(a_1,\ldots,a_n)$ such that $a_1,\ldots,a_n$ are different bound variables. Let Φ be the function given in Definition 1.5.1 and Ψ the function in Definition 1.5.2. Then $\mathit{Left}(s) = (\varnothing, s)^{\varnothing}_{\Phi_s}$, where $(\Delta, s)^{\Lambda}_{\Phi_s}$ is inductively defined as follows:

$$
\begin{array}{lcl}
(\Delta, a)^{\Lambda}_{\Phi_s} & = & (\Delta, a) \\
(\Delta, [a]t)^{\Lambda}_{\Phi_s} & = & (\Delta', [a]t'), \\
 & & \text{where } (\Delta', t') = (\Delta, t)^{\Lambda\cup\{a\}}_{\Phi_s} \\
(\Delta, ft)^{\Lambda}_{\Phi_s} & = & (\Delta', ft'), \\
 & & \text{where } (\Delta', t') = (\Delta, t)^{\Lambda}_{\Phi_s} \\
(\Delta, (t_1, \ldots, t_n))^{\Lambda}_{\Phi_s} & = & (\Delta', (t'_1, \ldots, t'_n)), \\
 & & \text{where } (\Delta, t_k)^{\Lambda}_{\Phi_s} = (\Delta_k, t'_k), \text{ for } 1 \leq k \leq n \\
 & & \text{and } \Delta' = \bigcup_k \Delta_k \\
(\Delta, Z_i^n(a_1, \ldots, a_n))^{\Lambda}_{\Phi_s} & = & (\Delta \cup \Delta', Z_i) \text{ if leftmost occurrence of } Z_i^n \text{ in } s \\
 & & \text{where } \Delta' = \{a\#Z_i \mid a \in \Lambda \setminus \Phi_s(Z_i^n)\} \\
(\Delta, Z_i^n(b_1, \ldots, b_n))^{\Lambda}_{\Phi_s} & = & (\Delta \cup \Delta', \Psi(\Phi_s(Z_i^n), [b_1, \ldots, b_n]) \cdot Z_i) \text{ otherwise,} \\
 & & \text{where } \Delta' = \{b\#Z_i \mid b \in \Lambda \setminus \Phi_s(Z_i^n)\}
\end{array}
$$

We use the notation for explicit atom substitution given in (Fernández et al., 2004), where $t[a \mapsto s]$ is an abbreviation for $\mathtt{sub}([a]t, s)$. We recall below the rules, then continue by formalising the definition of the right-hand side rule translation.

Definition 1.5.4 (Explicit Substitution Rules)**.** The following nominal rewrite rules define the behaviour of the binary function symbol `sub`. The notation $t[a \mapsto s]$ is syntactic sugar for $\mathtt{sub}([a]t, s)$

$$
\begin{array}{llcl}
(\sigma_{\mathtt{var}}) & a[a \mapsto X] & \to & X \\
(\sigma_{\varepsilon}) & a\#Y \vdash Y[a \mapsto X] & \to & Y \\
(\sigma_{f}) & (fX)[a \mapsto Y] & \to & fX[a \mapsto Y] \quad \text{for each } f \text{ in } \Sigma \\
(\sigma_{prod}) & (X_1, \ldots, X_n)[a \mapsto Y] & \to & (X_1[a \mapsto Y], \ldots, X_n[a \mapsto Y]) \\
(\sigma_{abs}) & b\#Y \vdash ([b]X)[a \mapsto Y] & \to & [b](X[a \mapsto Y])
\end{array}
$$

Definition 1.5.5 (Right Translation)**.** Let $l \Rightarrow r$ be a CRS rule. Let Φ_l be the function defined in Definition 1.5.1 applied to the CRS meta-term l. Then $Right(r) = (\Delta_r, [\![r]\!]_{\Phi_l})$ where

$$\Delta_r = \{a_k\#Z_i^n \mid a_k \text{ occurs bound above } Z_i^n \text{ in } r\}$$

and $[\![r]\!]_{\Phi_l}$ is defined by:

$$\begin{array}{lcl}
[\![a]\!]_{\Phi_l} & = & a \\
[\![fs]\!]_{\Phi_l} & = & f[\![s]\!]_{\Phi_l} \\
[\![[a]s]\!]_{\Phi_l} & = & [a][\![s]\!]_{\Phi_l} \\
[\![(t_1,\ldots,t_n)]\!]_{\Phi_l} & = & ([\![t_1]\!]_{\Phi_l},\ldots,[\![t_n]\!]_{\Phi_l}) \\
[\![Z_i^n(t_1,\ldots,t_n)]\!]_{\Phi_l} & = & \pi{\cdot}Z_i[\Phi_l(Z_i^n)_{m1} \mapsto [\![t_{m1}]\!]_{\Phi_l},\ldots,\Phi_l(Z_i^n)_{mk} \mapsto [\![t_{mk}]\!]_{\Phi_l}] \text{ where} \\
 & & \pi = (\Phi_l(Z_i^n)_{j1}\ [\![t_{j1}]\!]_{\Phi_l}){\cdot}\ldots{\cdot}(\Phi_l(Z_i^n)_{jk}\ [\![t_{jk}]\!]_{\Phi_l}), \text{ and} \\
 & & j1\ldots jk, m1\ldots mk \in \{1,\ldots,n\}, \\
 & & [\![t_{j1}]\!]_{\Phi_l},\ldots,[\![t_{jk}]\!]_{\Phi_l} \in A([\![(t_1,\ldots,t_n)]\!]_{\Phi_l})
\end{array}$$

Remark 1.5.6 (CRS Term Translation). For any CRS term t, t is also a nominal ground term, trivially, since there are no meta-variables.

Definition 1.5.7. We define the translation of the CRS rule $l \Rightarrow r$ as $\mathscr{C}^{\mathscr{R}}(l,r) = \Delta \vdash l' \to r'$, where $Left(l) = (\Delta_l, l')$, $Right(r) = (\Delta_r, r')$ and $\Delta = \Delta_l \cup \Delta_r$.

Definition 1.5.8 (Valuation translation)**.** Let t be a closed CRS meta-term, Φ_t as in Definition 1.5.1, and σ a safe valuation such that $\sigma = [Z_i^n \mapsto \underline{\lambda}(a_1,\ldots,a_n).s_i]$ for $1 \leq i \leq m$ where $dom(\sigma) \subseteq MV(t)$ and $\sigma(t)$ is ground. Then, $\langle\sigma\rangle_{\Phi_t} \triangleq [Z_i \mapsto \pi_i{\cdot}s_i]$ where $\pi_i = (a_n\ \Phi_t(Z_i^n)_n){\cdots}(a_1\ \Phi_t(Z_i^n)_1)$.

Below we denote by $(\nabla \vdash t', \sigma')$ the result of $(Left(t), \langle\sigma\rangle_{\Phi_t})$.

The pair of examples, Example 1.5.9 and Example 1.5.10, demonstrate that our translation function not only improves but also corrects the translation function it originated from in (Fernández et al., 2004).

Example 1.5.9 (Example 6.8 (Domínguez and Fernández, 2015))**.** The CRS meta-term $f([a]X,[b]X)$ is translated using *Left* as the closed nominal term

$$a\#X, b\#X \vdash f([a]X,[b]X)$$

whereas the original function from (Fernández et al., 2004) would translate the same term to $b\#X \vdash f([a]X,[b]X)$ which is not closed.

Additionally, thanks to the use of closed nominal rewriting, less freshness constraints are needed in the translation than when using (standard) nominal rewriting. However, in some cases freshness constraints are generated, even if a translation without freshnesses might be possible. For example, the CRS meta-term $f([a]X(a),[b]X(b))$ produces the closed nominal translation

$$b\#X \vdash f([a]X,[b](a\ b){\cdot}X)$$

where $b\#X$ ensures that the term is closed (see Definition 1.4.2); however, $\vdash f([a]X,[a]X)$ would also be a correct translation (note that $b\#X \vdash f([a]X,[b](a\ b)\cdot X) \approx_\alpha f([a]X,[a]X)$).

Example 1.5.10. The translation of the CRS meta-term

$$[c][a]f([c]Z(b,c,a),[d]Z(a,b,d))$$

to a closed NRS term using Definition 1.5.3 is

$$d\#Z \vdash [a][b]f([c]Z,[d](c\ d)(c\ a)(c\ b)\cdot Z).$$

A naive translation of the pair of argument lists into swappings would have modified the action of the permutation, that is, the solution $\pi = (b\ a)(c\ b)(a\ d)$ by the translation definition in Fernández et al. (2004) is not sound, since $\pi(b) = c$ and $\pi(c) = a$.

Definition 1.5.11 (σ-Normal Form of a Term-in-context)**.** We denote by $nf_\sigma(\Delta \vdash t)$ the normal form of the term-in-context $\Delta \vdash t$ with respect to the rules in Definition 1.5.4. It is uniquely defined modulo α-equivalence (Fernández et al., 2004).

Lemma 1.5.12 (Correctness of Explicit Substitution Rules)**.** *Let t, s be CRS terms (and therefore also nominal terms). Then $nf_\sigma(t[a \mapsto s]) \approx_\alpha t\{a \mapsto s\}$, where in the right-hand side $t\{a \mapsto s\}$ denotes the term obtained by substituting (using the capture-avoiding substitution of the CRS) a by s in t.*

Theorem 1.5.13 (Translation of CRS Rewrite Steps)**.** *Let $R = l \Rightarrow r$ be a CRS rule. Let u be a CRS term.*
If $u \Rightarrow_R v$ then $\vdash u \xrightarrow[\mathscr{R}\cup\sigma]{+}{}^{c} v$ using $\mathscr{R} = \mathscr{C}^{\mathscr{R}}(l,r)$ and the explicit substitution rules.

Part II

Extended Nominal Terms and α-equality

Chapter 2

Equality of Extended Nominal Terms. Derivability and Properties

This chapter aims to define a syntax for nominal terms extended with capture-avoiding substitution of atoms by terms in order to simulate higher-order substitution.

Substitution of atoms by terms is instrumental in the description of formal systems operating at object-level. Systems like lambda-calculus and predicate logic, for instance, contain object variables that can be bound over formulæ. Nominal algebra extended with atom substitution offers sufficient power to describe both quantification and instantiation over object variables, supported by common binders like λ, for the λ-calculus, ν for π-calculus and $\forall$ or $\exists$ for predicate logic.

We begin the chapter by introducing the syntax and grammar of extended nominal terms as well as extending the syntactic notions of position and occurrence to include the representation of atom substitution. Next, a notion of equality for extended nominal terms is defined in Definition 2.2.5 using the swapping operation and a freshness relation which has been extended to accommodate atom substitution in the derivation logic.

The action of atom substitutions is then presented in Definition 2.3.2 while demonstrating in Theorem 2.3.5 that the behaviour of atom substitution corresponds with the semantics of term-former `sub` as given by the set of explicit substitution rules in 1.5.4.

The chapter is concluded by demonstrating that the properties of standard nominal terms extend naturally to extended terms. Main claims are Theorem 2.5.8 and Theorem 2.5.19 showing α-equality to be an equality relation and that such relation is preserved under the application of atom actions, respectively.

2.1 Nominal Terms Extended with Atom Substitution

In this section we introduce the reader to the grammar of nominal terms extended with atom substitutions, describing common operations over the syntax of terms.

2.1.1 Syntax of extended nominal terms

A **nominal signature** Σ is a collection of **term-formers**, or **function symbols**, $f, g, \ldots$, each with a fixed arity. Fix a countably infinite set $\mathscr{X}$ of **variables** ranged over by $X, Y, Z, \ldots$, and a countably infinite set $\mathscr{A}$ of **atoms** ranged over by $a, b, c, \ldots$, and assume that Σ, $\mathscr{X}$ and $\mathscr{A}$ are pairwise disjoint.

The following signatures are inspired by examples given in (Fernández and Gabbay, 2005; Fernández et al., 2004; van Raamsdonk, 2015). These signatures will be applied to recurrent examples during this and future chapters.

Example 2.1.1. The nominal signature for the standard higher-order function *map* has function symbols

$$\texttt{nil}:0 \quad \texttt{cons}:2 \quad \texttt{map}:2.$$

where notation _:_ relates the function symbol to its fixed arity.

The signature for summation contains function symbols

$$\texttt{0}:0 \quad \texttt{suc}:2 \quad \Sigma:2 \quad \texttt{g}:2.$$

The signature for the differentiation operator contains the following function symbols

$$\times:2 \quad \texttt{diff}:2 \quad \texttt{sin}:1 \quad \texttt{cos}:1.$$

Finally, the asynchronous π-calculus has a signature

$$\texttt{in}:2 \quad \texttt{out}:2 \quad \texttt{rep}:1 \quad \texttt{par}:2 \quad \nu:1.$$

Permutations on atoms were already present in non-extended nominal terms. We briefly recall their definition below.

Definition 2.1.2 (Permutation of atoms)**.** A **permutation** π is a bijection on $\mathscr{A}$ such that the set of atoms for which $a \neq \pi(a)$ is finite. This set is called the **support** of π, written $\mathscr{S}upport(\pi)$.

A **swapping** is a pair of atoms, written $(a\ b)$, where a maps to b, b maps to a and all other c map to themselves. Then, a permutation π is represented by a list of swappings applied in

the right-to-left order and generated by the grammar $\pi ::= \mathtt{Id} \mid \pi(a\ b)$, where $\mathtt{Id}$ denotes the **identity permutation** and is commonly omitted.

Write $\pi' \circ \pi$ for **composition** of permutations such that $(\pi' \circ \pi)(a) = \pi'(\pi(a))$ and write π^{-1} for the **inverse** of π, that is, if $\pi = (a\ b)(b\ c)$ then $\pi(c) = a$ if and only if $c = \pi^{-1}(a)$.

We are ready to introduce the grammar of nominal terms with atom substitution. The syntax is based on (Fernández and Gabbay, 2007; Urban et al., 2004), with minor alterations to the syntax of extended terms previously presented in (Fairweather et al., 2015).

Definition 2.1.3. Extended nominal terms, or just **terms**, are generated by the grammar

$$s,t,l,r ::= a \mid \phi\hat{}\pi\cdot X \mid [a]s \mid fs \mid (s_1,\ldots,s_n)$$

and called, respectively, **atoms**, **moderated variables** or simply **variables**, **abstractions**, **function applications** (which must respect the arity of the function symbol) and **tuples** (if $n = 0$ or $n = 1$ we may omit the parentheses). A moderated variable is $X \in \mathscr{X}$ along with a **suspended** permutation π and followed by a suspended atom substitution ϕ, as given in Definition 2.1.5. An abstraction $[a]t$ is intended to represent t with a bound; we say that **the scope of** abstractor $[a]$- is t. Then, call occurrences of a **abstracted** if they are in the scope of an *abstractor* for a, and **unabstracted**, or free, otherwise.

Example 2.1.4 (Abstraction term)**.** The term $[\mathtt{a}](\mathtt{a},\mathtt{b})$ has atom $\mathtt{a}$ abstracted and atom $\mathtt{b}$ unabstracted. Both atoms are in the scope of $[\mathtt{a}]$-.

Definition 2.1.5. Atom substitutions ϕ, or just **a-substitutions**, are mappings from atoms to terms with finite domain, generated by the grammar $\phi ::= \mathtt{Id} \mid [a \mapsto s]\phi$. The final $\mathtt{Id}$ is commonly omitted.

The **image of an atom** a **under an a-substitution** ϕ is denoted by $\phi(a)$. Then, the **domain** of ϕ, written $\mathscr{D}om(\phi)$, is the finite set of atoms such that $\phi(a) \neq a$. Write $\phi^{-a_1,\ldots,a_n}$ for the a-substitution ϕ with domain $\mathscr{D}om(\phi) \setminus \{a_1,\ldots,a_n\}$. The **image** of ϕ, written $\mathscr{I}mg(\phi)$, is the set of terms $\{\phi(a) \mid a \in \mathscr{D}om(\phi)\}$.

Composition of atom substitutions, written $\phi \bullet \phi'$, is applied inversely since the notation is postfix. Hence $s(\phi \bullet \phi') = (s\phi)\phi'$. An alternative approach, merging both mappings, is defined later in Lemma 2.3.4.

Although a-substitutions ϕ are represented as a sequential list of bindings of form $[a_1 \mapsto s_1]\cdots[a_n \mapsto s_n]$, they are, in fact, interpreted as a set of *simultaneous* bindings where the value of ϕ when applied to any $a_i \in \mathscr{D}om(\phi)$ is defined directly from the image term of a_i, $\phi(a_i)$. Intuitively, it means that the application of a-substitutions ϕ to a term simultaneously replaces,

in a *capture-avoidance* manner, all occurrences of atoms by their respective ϕ-images. An axiomatisation of the operation is postponed until Section 2.3 (see Definition 2.3.2), after providing a formal definition of logical equivalence for nominal terms decorated with a-substitutions. Then, we are able to describe a mechanism to avoid capture of atoms when a-substitutions act on terms.

In the sequel, a sequential list of simultaneous a-substitution mappings $[a_1 \mapsto s_1] \cdots [a_n \mapsto s_n]$ is represented as $[a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]$.

We abbreviate $\mathtt{Id}\hat{}\pi \cdot X$ as $\pi \cdot X$, $\phi\hat{}\mathtt{Id} \cdot X$ as $\phi \cdot X$ and also $\mathtt{Id}\hat{}\mathtt{Id} \cdot X$ as X if there is no ambiguity.

Example 2.1.6 (Extended nominal terms)**.** We write some extended nominal terms using the signatures mentioned in Example 1.2.1 and Example 2.1.1.

$\mathtt{app}(\mathtt{abs}([\mathtt{a}]\mathtt{X}), \mathtt{Y})$	$\mathtt{app}(\mathtt{abs}([\mathtt{a}]\mathtt{app}(\mathtt{a},\mathtt{a})), \mathtt{abs}([\mathtt{a}]\mathtt{app}(\mathtt{a},\mathtt{a})))$
$\mathtt{app}(\mathtt{abs}([\mathtt{c}][\mathtt{a} \mapsto \mathtt{abs}([\mathtt{n}]\mathtt{n})]\hat{}(\mathtt{b}\ \mathtt{c}) \cdot \mathtt{X}), \mathtt{Y})$	$\mathtt{g}(\Sigma(\mathtt{N}, [\mathtt{a}]\mathtt{F}), [\mathtt{a} \mapsto \mathtt{suc}(\mathtt{N})] \cdot \mathtt{F})$
$\mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}] \cdot \mathtt{F}), \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{T})$	$\mathtt{par}(\mathtt{out}(\mathtt{a},\mathtt{b}), \mathtt{in}(\mathtt{a}, [\mathtt{c}]\mathtt{P}))$.

For more examples, we refer the reader to (Fernández and Gabbay, 2007; Urban et al., 2004) for (non-extended) nominal terms and (Fairweather et al., 2015) for the extended case.

In the sequel, we sometimes refer to permutations and atom substitutions together as **atom actions**. Further, if we want to be more specific we say *the **atom actions of** a variable* X to refer to the pair of permutation and atom substitution suspended over an occurrence of variable X, when the occurrence is clear to us. Then, call **domain of the atom actions of** X the union of the domain of the atom substitution and the support of the permutation suspended over the same occurrence of X. Additionally, nominal terms as introduced in (Urban et al., 2004) will be referred to as *non-extended* or *standard* (nominal) terms, to distinguish them from nominal terms with atom substitution introduced in this chapter.

We are now ready to extend the action of permutations to terms.

2.1.2 Permutation action

Definition 2.1.7. The **action of a permutation π on a-substitutions** is defined as

$$\pi \cdot ([a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]) \triangleq [\pi(a_1) \mapsto \pi \cdot s_1; \ldots; \pi(a_n) \mapsto \pi \cdot s_n]$$

where $\pi \cdot s_i$ is the application of π to term s_i , for $(1 \leq i \leq n)$, as Defined in 2.1.8.

Definition 2.1.8. The **action of a permutation π on a term** t, written $\pi{\cdot}t$, is defined by induction on terms as follows:

$$\begin{array}{ll} \mathtt{Id}{\cdot}t \triangleq t \qquad \pi{\cdot}a \triangleq \pi(a) & (a\ b)\pi{\cdot}t \triangleq (a\ b){\cdot}(\pi{\cdot}t) \\ \pi{\cdot}[a]t \triangleq [\pi(a)]\pi{\cdot}t & \pi{\cdot}(\phi\hat{}\pi'{\cdot}X) \triangleq (\pi{\cdot}\phi)\hat{}(\pi\circ\pi'){\cdot}X \\ \pi{\cdot}ft \triangleq f\pi{\cdot}t & \pi{\cdot}(t_1,\ldots,t_n) \triangleq (\pi{\cdot}t_1,\ldots,\pi{\cdot}t_n). \end{array}$$

Example 2.1.9 (Permutation action)**.**
$(\mathtt{a}\ \mathtt{b}){\cdot}\mathtt{f}(\mathtt{c},[\mathtt{a}\mapsto[\mathtt{b}]\mathtt{b}]\hat{}(\mathtt{b}\ \mathtt{c}){\cdot}\mathtt{X}) = \mathtt{f}(\mathtt{c},[\mathtt{b}\mapsto[\mathtt{a}]\mathtt{a}]\hat{}(\mathtt{a}\ \mathtt{b})(\mathtt{b}\ \mathtt{c}){\cdot}\mathtt{X})$.

2.1.3 Positions and occurrence of terms

Definition 2.1.10. Functions $V(t)$ and $A(t)$ are used to compute the sets of variables and atoms, respectively, appearing in a term t. They are inductively defined as follows:

$$V(a) \triangleq \varnothing \qquad V(\phi\hat{}\pi{\cdot}X) \triangleq \bigcup_{a_i\in\mathscr{D}om(\phi)} V(\phi(a_i))\cup\{X\}$$
$$V([a]t) \triangleq V(t) \qquad V(fs) \triangleq V(s) \qquad V((s_1,\ldots,s_n)) \triangleq V(s_1)\cup\ldots\cup V(s_n)$$

$$A(a) \triangleq \{a\} \qquad A(\phi\hat{}\pi{\cdot}X) \triangleq \bigcup_{a_i\in\mathscr{D}om(\phi)} A(\phi(a_i))\cup\mathscr{D}om(\phi)\cup\mathscr{S}upport(\pi)$$
$$A([a]t) \triangleq A(t)\cup\{a\} \qquad A(fs) \triangleq A(s) \qquad A((s_1,\ldots,s_n)) \triangleq A(s_1)\cup\ldots\cup A(s_n)$$

Ground terms have no variables: $V(t)=\varnothing$.

Outermost brackets are commonly omitted when $V(\cdot)$ (resp. $A(\cdot)$) is applied to a tuple, that is, $V((s_1,\ldots,s_n))$ (resp. $A((s_1,\ldots,s_n))$) is thus denoted as $V(s_1,\ldots,s_n)$ (resp. $A(s_1,\ldots,s_n)$). We sometimes abuse the notation of function $V(\cdot)$ (resp. $A(\cdot)$) to compute the set of variables (resp. atoms) in the image of an a-substitution $\mathscr{I}mg(\phi)$, namely $V(\mathscr{I}mg(\phi))$ (resp. $A(\mathscr{I}mg(\phi))$). Then, set $\mathscr{I}mg(\phi)$ should be thought of as a tuple.

Sometimes a distinction must be made between variables occurring in suspended a-substitutions and variables which do not. When such distinction is required, we call the former **suspended (variable) occurrence** and the latter **fixed (variable) occurrence**. The set of fixed variable occurrences in a term is obtained as follows.

Definition 2.1.11. The set of variable symbols with at least one fixed occurrence having suspended trivial a-substitutions in a term t is denoted by $V_f(t)$ and inductively defined as in function $V(t)$ (see Definition 2.1.10) except equality $V(\phi\hat{}\pi{\cdot}X) \triangleq \bigcup_{a_i\in\mathscr{D}om(\phi)} V(\phi(a_i))\cup\{X\}$ is now defined as

$$V_f(\phi\hat{}\pi\cdot X) \triangleq \begin{cases} \{X\} & \text{if } \phi = \mathtt{Id}; \\ \varnothing & \text{otherwise} \end{cases}$$

Some variable symbols may have suspended and fixed occurrences in the same term. We are interested on distinguishing variable symbols which have fixed occurrences and where at least one of the occurrences has trivial a-substitutions suspended over such variable symbol. This set of variable symbols will play an important part in subsequent chapters. First in Chapter 5, when restrictions to the structure of terms are applied during the definition of a *unitary matching algorithm* and again in Chapter 6 when describing the class of *extended nominal rewrite rules*.

Example 2.1.12 (Sets of atoms and variables in a term)**.** The term $\mathtt{t} = [\mathtt{a}][\mathtt{b} \mapsto (\mathtt{X},\mathtt{c})]\hat{}(\mathtt{d}\ \mathtt{e})\cdot\mathtt{Y}$ has $\mathtt{A}(\mathtt{t}) = \{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e}\}$ and $\mathtt{V}(\mathtt{t}) = \{\mathtt{X},\mathtt{Y}\}$. Also, the set of variable symbols with fixed occurrences where a-substitution is trivial is, $V_f(\mathtt{t}) = \varnothing$.

The following definitions over the syntax of nominal terms provides a valuable insight on the structure of extended nominal terms. An arbitrary ordering is chosen when defining the positioning of the terms in the image of a-substitutions (see Remark 2.1.14).

Definition 2.1.13 (Positions and subterms of extended terms)**.** Let s be a nominal term, then the set of **positions** of s is a set $\mathscr{P}os(s)$ of strings of positive integers, inductively defined below. We also define below the subterms of s: call $s|_p$ a **subterm** of s at position p.

- if $s = a$, then $\mathscr{P}os(s) \triangleq \{\varepsilon\}$ and $s|_\varepsilon = s$, where ε denotes the empty string;
- if $s = [a_1 \mapsto t_1; \cdots; a_n \mapsto t_n]\hat{}\pi\cdot X$, then $\mathscr{P}os(s) \triangleq \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i\cdot p \mid p \in \mathscr{P}os(t_i)\}$, $s|_\varepsilon = s$ and $s|_{i\cdot p} = t_i|_p$;
- if $s = [a]t$, then $\mathscr{P}os(s) \triangleq \{\varepsilon\} \cup \{1\cdot p \mid p \in \mathscr{P}os(t)\}$, $s|_\varepsilon = s$ and $s|_{1\cdot p} = t|_p$;
- if $s = ft$, then $\mathscr{P}os(s) \triangleq \{\varepsilon\} \cup \{1\cdot p \mid p \in \mathscr{P}os(t)\}$, $s|_\varepsilon = s$ and $s|_{1\cdot p} = t|_p$;
- if $s = (t_1,\ldots,t_n)$, then $\mathscr{P}os(s) \triangleq \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i\cdot p \mid p \in \mathscr{P}os(t_i)\}$, $s|_\varepsilon = s$ and $s|_{i\cdot p} = t_i|_p$.

The position ε is called **the root position of** the term s, and the symbol at this position is called the **root symbol** of s. The **prefix order** is a partial order on positions defined as $p \leq q$ *if and only if* there is a p' such that $pp' = q$. Then, we say that position p is **above** position q, or position p is **strictly above** position q if $p < q$. Otherwise, we say that positions p, q are **parallel** ($p \parallel q$), that is, they are incomparable with respect to $\leq$.

The size of a term t, $|t|$, is the cardinality of the set $\mathscr{P}os(t)$.

If $p \in \mathscr{P}os(s)$, then $s[t]_p$ denotes the term that is obtained from s by **replacing the subterm at position p by** term t. When p, q are two parallel positions in s, $p \parallel q$, $(s[t]_p)[t']_q = (s[t']_q)[t]_p$ for any pair of terms t, t'. Sometimes $(\cdots(s[t_1]_{p_1})\cdots)[t_n]_{p_n}$ is denoted as $s[t_1 \cdots t_n]_{p_1 \cdots p_n}$ when $p_1 \parallel \cdots \parallel p_n$, where $s, t_1 \cdots t_n$ are terms and $p_1 \cdots p_n \in \mathscr{P}os(s)$. We refer to any nominal term u that is the same as another term t everywhere except below a position p as the **context** within which a replacement takes place, namely, if $t|_p = s$ then $u[s]_p = t$. Therefore, a context is a nominal term u with a distinguished position p.

In the sequel, we may sometimes say *variables at a fixed position* to denote fixed variable occurrences. Similarly we may say *variables at a suspended position* to denote suspended variable occurrences.

The following remark clarifies on the description of set of positions for moderated variables.

Remark 2.1.14 (Positions of a-substitutions). We have used an arbitrary positioning for the set of terms in the image of a-substitutions in order to describe the structure of an extended nominal syntax tree. The chosen positioning replicates the syntactic representation of a-substitutions as a lexicographic sequence with respect to the atoms in its domain. However, since the action of a-substitutions is simultaneous, nominal syntax trees differing only on the positioning of a-substitutions suspended directly below the same variable occurrence have the same properties; this becomes clearer when defining α-equivalence of extended terms (see Definition 2.2.5 and Definition 2.4.4). Then, we observe that variable occurrences differing only on the positioning of a-substitutions as described above are logically equivalent as a result of their *disagreement set* being empty (see rule $({}_?{\approx}_?{}_{\mathbf{alt}}\mathbf{X})$). We omit the inductive proof.

Positions are sometimes called *occurrences*, however, we use this denomination to refer to a subterm $t|_p$ instead. Then,

Definition 2.1.15 (Term occurrence)**.** Say a term s **occurs in (the syntax of)** a term t, written $s \lhd t$, when $t|_p = s$ for some position p. Then, write $s \not\lhd t$ for the case where s does not occur in the syntax of term t.

We extend the definition to constraints (see Definition 2.2.1 for constraint grammar) such that:

- if $C = s \approx_\alpha t$ we say that s (resp. t) **occurs in (the syntax of)** C, $s \lhd C$ (resp. $t \lhd C$);
- if $C = a\#s$ we say that a (resp. s) **occurs in (the syntax of)** C, $a \lhd C$ (resp. $s \lhd C$).

We end the section with an example that shows the positions and subterms of a term.

Example 2.1.16 (Set of positions and subterms of a term)**.** The term $\mathtt{s} = \mathtt{f}([\mathtt{a}]\mathtt{c}, \mathtt{b}, [\mathtt{n} \mapsto \mathtt{Y}]\widehat{}(\mathtt{c}\ \mathtt{d}) \cdot \mathtt{X})$ has the following set of positions: $\mathscr{P}os(\mathtt{s}) = \{\varepsilon, 1, 11, 111, 12, 13, 131\}$, thus $|s| = 7$.

Then, $\mathtt{f}([\mathtt{a}]\mathtt{c},\mathtt{b},[\mathtt{n}\mapsto\mathtt{Y}]\widehat{\ }(\mathtt{c}\ \mathtt{d})\cdot\mathtt{X}) = \mathtt{s}|_{\varepsilon}$, $([\mathtt{a}]\mathtt{c},\mathtt{b},[\mathtt{n}\mapsto\mathtt{Y}]\widehat{\ }(\mathtt{c}\ \mathtt{d})\cdot\mathtt{X}) = \mathtt{s}|_{1}$, $[\mathtt{a}]\mathtt{c} = \mathtt{s}|_{11}$, $\mathtt{c} = \mathtt{s}|_{111}$, $\mathtt{b} = \mathtt{s}|_{12}$, $[\mathtt{n}\mapsto\mathtt{Y}]\widehat{\ }(\mathtt{c}\ \mathtt{d})\cdot\mathtt{X} = \mathtt{s}|_{13}$ and $\mathtt{Y} = \mathtt{s}|_{131}$.

Also, we have $\mathtt{f}([\mathtt{a}]\mathtt{c},\mathtt{b},[\mathtt{n}\mapsto\mathtt{Y}]\widehat{\ }(\mathtt{c}\ \mathtt{d})\cdot\mathtt{X}) \lhd \mathtt{s}$, $\mathtt{c}\lhd\mathtt{s}$, $(\mathtt{c}\ \mathtt{d})\cdot\mathtt{X}\lhd\mathtt{s}$ and $\mathtt{Y}\lhd\mathtt{s}$ as well as $1 < 11, 1 < 12, 13 < 131$ but $11 \parallel 12$ and $12 \parallel 131$, for instance.

Write $\mathtt{a}\lhd\mathtt{a}\#[\mathtt{a}]\mathtt{b}$ and $[\mathtt{a}]\mathtt{b}\lhd\mathtt{a}\#[\mathtt{a}]\mathtt{b}$.

In the next section, we provide a formal definition of logical equality between nominal terms in the presence of a-substitutions.

2.2 Freshness and α-Equivalence. A Logical Presentation

In this section we extend the notions of *freshness* and α*-equality* previously introduced for standard nominal terms, to incorporate a-substitutions. A direct consequence of such inclusion is an increment in the complexity of the set of *inference rules* for both relations, freshness and α-equality; this is due to terms in the image of a-substitutions occurring suspended on variables. Then, to claim that the freshness (resp. α-equality) relation holds for an atom and a variable occurrence (resp. a pair of variable occurrences sharing the same variable symbol) involves an inductive verification of *derivability* for the relation with respect to each term in the image of a-substitutions.

Definition 2.2.1 (Freshness and α-equality constraints)**.** Constraints for extended nominal terms are generated by the grammar:

$$C ::= a\#t \mid s \approx_{\alpha} t \mid \top \mid \bot.$$

Call symbol # a **freshness** predicate and symbol $\approx_{\alpha}$ an **α-equivalence**, or **α-equality**, predicate. Their interpretation and formal definition are given below. Also, symbols $\top$ and $\bot$ denote **true** and **failure** constraints, respectively.

We present a set of *deterministic, syntax-directed natural deduction* rules to describe the relation on extended terms determined by predicates # and $\approx_{\alpha}$. Following previous descriptions of the derivation rules (see (Urban et al., 2004), for instance), we apply a *sequent calculus* style (Goubault-Larrecq and Mackie, 2001) for the representation of the set of derivation rules, finding the style more suitable to accommodate the requirements of the extended framework, as we show next.

This chapter offers two equivalent versions of both freshness and α-equality inference rules, known as *core* and *alternative* inference rules. Core freshness and core α-equality inference rules are introduced first since both are more suitable for inductive definition and a

direct transformation into algorithmic form (see Chapter 3), whereas alternative freshness and alternative α-equality inference rules will be used for the definition of theorems and description of proofs along this thesis. Alternative inference rules and the equivalence proofs between both versions of inference rules are introduced later, in Section 2.4.

2.2.1 Core freshness judgements

We begin the representation by introducing the notion of freshness on extended nominal terms.

Definition 2.2.2 (Freshness constraint)**.** A **freshness constraint** is a pair $a\#t$ of an atom and a term. Intuitively, $a\#t$ means that if a occurs in t, it must be in the scope of an abstractor for a. A **freshness context** (ranged over by $\Delta, \nabla, \Delta', \nabla'$ and so on) is a set of **primitive constraints** of the form $a\#X$.

So now we have two types of contexts, terms with a distinguished position as given in Definition 2.1.13 and sets of freshness constraints as described above. There will be no ambiguity of their intended meaning because of the circumstances in which each of them will be used. Also, we always specify that it is a freshness context when referring to the set of freshness constraints.

Freshness contexts play an active role, providing *evidence*, in the form of primitive constraints, for a freshness relation to hold in the presence of variables. Primitive constraints are assumptions on variables, explicitly denoting freshness conditions that must be preserved for specific atoms when replacing variables by terms. Recall that a freshness relation between an atom and a standard moderated variable is satisfied simply by finding evidence of such relation in the freshness context. For the extended variable case, satisfiability of the freshness relation depends also on the relation being consistent for each term in the image of a-substitutions, specifically by providing a derivation proof of preservation of the freshness judgement on each term in the image of a-substitutions or, by ensuring that variables cannot be replaced by terms containing occurrences of unabstracted atoms that would trigger the action of those a-substitutions not meeting the freshness requirements.

Derivability of freshness constraints is recursively determined by the set of rules and axioms specified below.

Definition 2.2.3 (Core freshness rules)**.** Write $\nabla \vdash a\#t$ when a derivation exists using the rules below. Then, say that ∇ **entails** $a\#t$ or that $a\#t$ **is derivable from** ∇, and call $\nabla \vdash a\#t$ a

core freshness judgement.

$$\frac{}{\nabla \vdash a\#b}\ (\#\mathbf{ab}) \qquad \frac{}{\nabla \vdash a\#[a]s}\ (\#[\mathbf{a}]) \qquad \frac{\nabla \vdash a\#s}{\nabla \vdash a\#[b]s}\ (\#[\mathbf{b}]) \qquad \frac{\nabla \vdash a\#s}{\nabla \vdash a\#fs}\ (\#\mathbf{f})$$

$$\frac{\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} (\nabla \vdash a\#\phi(b) \vee (\pi^{-1}(b)\#X \in \nabla))}{\nabla \vdash a\#\phi\hat{}\pi\cdot X}\ (\#\mathbf{X}) \qquad \frac{\nabla \vdash a\#s_1 \cdots \nabla \vdash a\#s_n}{\nabla \vdash a\#(s_1,\ldots,s_n)}\ (\#\mathbf{tupl})$$

We may drop set brackets in freshness contexts, eg. $a\#X, b\#Y$ for $\{a\#X, b\#Y\}$. Also, sometimes we write $a, b\#t$ (resp. $a\#s,t$) instead of $a\#t, b\#t$ (resp. $a\#s, a\#t$). Commas are used as logical connectives on both sides of the turnstile: $a\#X, b\#Y \vdash a\#s, b\#t$. A comma should be thought of as an *and*.

The set of core freshness rules remains similar to previous definitions for both non-extended (see (Fernández and Gabbay, 2007; Urban et al., 2004), for instance) and extended (see (Fairweather et al., 2015)) nominal terms, except for the extended core variable rule (#**X**). Rule (#**X**) broadens the non-extended derivation rule for variables by describing the necessary conditions for a core freshness judgement of form $\nabla \vdash a\#\phi\hat{}\pi\cdot X$ to hold when a-substitutions appear suspended on variables. Informally, $a\#\phi\hat{}\pi\cdot X$ holds when $a\#\phi(b)$ is derivable from ∇, hence $\nabla \vdash a\#\phi(b)$, or when $\pi^{-1}(b)\#X$ is an assumption in ∇, for each atom b in the domain of ϕ. This ensures that atom a will not be introduced by any term in the image of ϕ when applying an instance of X and it is shown in the premise of core rule (#**X**) as a conjunctive normal form (CNF) with each conjunct being a disjunction of form $a\#\phi(b) \vee \pi^{-1}(b)\#X$ for every $b \in \mathscr{D}om(\phi)$. Additionally, atom a must also be included along with the atoms in the domain of ϕ. Assume atom a is not a member of $\mathscr{D}om(\phi)$, then $\phi(a) = a$ and we observe that core judgement $\nabla \vdash a\#\phi(a)$ fails as expected, however, for this case the interest is on asserting that the permutation action does not introduce atom a by means of a swap, namely that evidence for assumption $\pi^{-1}(a)\#X$ exists in ∇. This case is indeed the derivation case for variables in standard nominal terms as seen in (Fernández and Gabbay, 2007; Urban et al., 2004), among others, and it leads to an interesting property of the system above described: *for the case where a-substitutions are* `Id`*, derivation of the variable rule is reduced to derivation of its analogue as given in (Fernández and Gabbay, 2007, Definition 6) for standard nominal terms*.

Our notation of core rule (#**X**) varies with respect to the one given in (Fairweather et al., 2015) in that (Fairweather et al., 2015) has two inference rules for the variable case such that

one rule deals specifically with the case where, following the notation above, atom a is not in the domain of ϕ and therefore $\pi^{-1}(a)\#X \in \nabla$ must hold.

The freshness relation $a\#s$ is defined by induction on the size of s and can be seen as a decidable and syntax-directed finite recursive predicate. The only case that is not straight-forward is that of a variable, however, observe that the premise of (#**X**) is indeed syntax-directed and structurally of smaller size than its conclusion since atom actions are represented as finite mappings and the image of a-substitutions occurs as subterms in the syntax of variables (see Definition 2.1.13).

We have described a set of core freshness derivation rules where, due to rule (#**X**), there may exist more than one derivation path from a given freshness context. As a result, there may also exist more than one least set of primitive constraints logically entailing the derivation of the same core freshness relation. This feature differs from non-extended nominal terms where the freshness relation is derived from a unique least set of assumptions. An example follows.

Example 2.2.4. Freshness constraint $\mathtt{a}\#[\mathtt{b} \mapsto \mathtt{Y}]\hat{}(\mathtt{a}\ \mathtt{c})\cdot\mathtt{X}$ is derivable from $\nabla_1 = \{\mathtt{a}\#\mathtt{Y}, \mathtt{c}\#\mathtt{X}\}$ or from $\nabla_2 = \{\mathtt{b}\#\mathtt{X}, \mathtt{c}\#\mathtt{X}\}$ using rule (#**X**) from Definition 2.2.3 as follows.

- From context ∇_1 and $\mathscr{D}\mathtt{om}(\phi) \cup \{\mathtt{a}\} = \{\mathtt{a}, \mathtt{b}\}$ we have

$$\frac{\mathtt{c}\#\mathtt{X} \in \nabla_1 \wedge \nabla_1 \vdash \mathtt{a}\#\mathtt{Y}}{\nabla_1 \vdash \mathtt{a}\#[\mathtt{b} \mapsto \mathtt{Y}]\hat{}(\mathtt{a}\ \mathtt{c})\cdot\mathtt{X}}\ (\#\mathbf{X})$$

- From context ∇_2 and $\mathscr{D}\mathtt{om}(\phi) \cup \{\mathtt{a}\} = \{\mathtt{a}, \mathtt{b}\}$ we have

$$\frac{\mathtt{c}\#\mathtt{X} \in \nabla_2 \wedge \mathtt{b}\#\mathtt{X} \in \nabla_2}{\nabla_2 \vdash \mathtt{a}\#[\mathtt{b} \mapsto \mathtt{Y}]\hat{}(\mathtt{a}\ \mathtt{c})\cdot\mathtt{X}}\ (\#\mathbf{X})$$

Next, we define logical equivalence for extended nominal terms.

2.2.2 Core α-equality judgements

In the sequel, for the sake of brevity, we denote $\mathscr{D}om(\phi) \cup \mathscr{D}om(\phi')$ by $\mathscr{D}omP(\phi, \phi')$ and $\mathscr{S}upport(\pi) \cup \mathscr{S}upport(\pi')$ by $\mathscr{S}upportP(\pi, \pi')$, for any pair of a-substitutions ϕ, ϕ' and permutations π, π'. Then, $\mathscr{D}om(\phi) \cup \mathscr{D}om(\phi') \cup \mathscr{S}upport(\pi) \cup \mathscr{S}upport(\pi') = \mathscr{D}omP(\phi, \phi') \cup \mathscr{S}upportP(\pi, \pi')$.

Definition 2.2.5 (Core α-equivalence rules)**.** An **α-equivalence constraint** is a pair $s \approx_\alpha t$ of terms. Write $\nabla \vdash s \approx_\alpha t$ when a derivation exists using the core rules below. Then, say

that ∇ **entails** $s \approx_\alpha t$ or that $s \approx_\alpha t$ **is derivable from** ∇. Call this a *core* ***α-equivalence judgement***.

$$\frac{}{\nabla \vdash a \approx_\alpha a}\ (\approx_\alpha \mathbf{a}) \qquad \frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash fs \approx_\alpha ft}\ (\approx_\alpha \mathbf{f}) \qquad \frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash [a]s \approx_\alpha [a]t}\ (\approx_\alpha [\mathbf{a}])$$

$$\frac{\bigwedge_{a \in (\mathscr{D}omP(\phi,\phi') \cup \mathscr{S}upportP(\pi,\pi'))} ((\nabla \vdash \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))) \vee a\#X \in \nabla)}{\nabla \vdash \phi\hat{}\pi \cdot X \approx_\alpha \phi'\hat{}\pi' \cdot X}\ (\approx_\alpha \mathbf{X})$$

$$\frac{\nabla \vdash (b\ a)\cdot s \approx_\alpha t \quad \nabla \vdash b\#s}{\nabla \vdash [a]s \approx_\alpha [b]t}\ (\approx_\alpha [\mathbf{b}]) \qquad \frac{\nabla \vdash s_1 \approx_\alpha t_1 \cdots \nabla \vdash s_n \approx_\alpha t_n}{\nabla \vdash (s_1,\ldots,s_n) \approx_\alpha (t_1,\ldots,t_n)}\ (\approx_\alpha \mathbf{tupl})$$

Our presentation of alpha-equality is a generalisation of the notion of alpha-equivalence for non-extended terms, defined by induction on the size of the pair of terms (s,t) for any predicate of form $s \approx_\alpha t$, and can be seen as a decidable and syntax-directed recursive predicate. Similarly to the freshness relation, the variable case needs further clarification. A variable occurrence has atom actions as finite mappings and the image of suspended a-substitutions are given as subterms of the syntax of a variable. Then, $\phi\hat{}\pi \cdot X \approx_\alpha \phi'\hat{}\pi' \cdot X$ is derivable from some freshness context ∇ if there is an assumption $a\#X$ in ∇ or $\phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))$ is derivable from ∇, for each atom a occurring in the domain of ϕ or ϕ', $\mathscr{D}omP(\phi,\phi')$, or in the support of π or π', $\mathscr{S}upportP(\pi,\pi')$. Therefore, the premise of $(\approx_\alpha \mathbf{X})$ can be seen as a finite recursive predicate which is both syntax-directed and of smaller size that its conclusion.

Recall that the premise in $(\approx_\alpha \mathbf{X})$ for standard nominal terms (see (Urban et al., 2004)) is a conjunction of primitive constraints between a given variable X and the set of atoms in the support of π or π' for which application of each permutation does not resolve to the same atom. This set of atoms is commonly known as the *disagreement*, or *difference, set* of the pair of permutations suspended on both occurrences of variable X. In our generalised version of $(\approx_\alpha \mathbf{X})$, the premise is a conjunction of disjunctions of primitive constraints and core alpha-equality judgements because of subterms possibly occurring suspended over both variable occurrences. This definition of alpha-equivalence for variables follows that given in Fairweather et al. (2015), however, here it is given as a CNF instead of the disagreement set notation; our definition is more suitable for a direct conversion to algorithmic form, as we show in Chapter 3. An alternative notation, using a difference set for extended terms, is given in Section 2.4.

Unlike standard nominal terms (Urban et al., 2004), there may exist more than one least set of primitive constraints logically entailing the derivation of an α-equivalence relation between distinct occurrences of the same variable, as we demonstrate next.

Example 2.2.6. The α-equality constraint $[\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b} \mapsto \mathtt{Y}]\cdot(\mathtt{b}\ \mathtt{c})\cdot\mathtt{X}$ is derivable from $\nabla_1 = \{\mathtt{b\#X}, \mathtt{a\#Y}, \mathtt{b\#Y}\}$ or from $\nabla_2 = \{\mathtt{b\#X}, \mathtt{c\#X}\}$, using rule $(\approx_\alpha \mathbf{X})$ from Definition 2.2.5 as follows.

- From context ∇_1 and $\mathscr{D}\mathtt{omP}([\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}], [\mathtt{b} \mapsto \mathtt{Y}]) \cup \mathscr{S}\mathtt{upportP}(\mathtt{Id}, (\mathtt{b}\ \mathtt{c})) = \{\mathtt{b}, \mathtt{c}\}$ we have
$$\frac{\mathtt{b\#X} \in \nabla_1 \wedge \nabla_1 \vdash (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y} \approx_\alpha \mathtt{Y}}{\nabla_1 \vdash [\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b} \mapsto \mathtt{Y}]\cdot(\mathtt{b}\ \mathtt{c})\cdot\mathtt{X}}\ (\approx_\alpha \mathbf{X})$$
- From context ∇_2 and $\mathscr{D}\mathtt{omP}([\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}], [\mathtt{b} \mapsto \mathtt{Y}]) \cup \mathscr{S}\mathtt{upportP}(\mathtt{Id}, (\mathtt{b}\ \mathtt{c})) = \{\mathtt{b}, \mathtt{c}\}$ we have
$$\frac{\mathtt{b\#X} \in \nabla_2 \wedge \mathtt{c\#X} \in \nabla_2}{\nabla_2 \vdash [\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b} \mapsto \mathtt{Y}]\cdot(\mathtt{b}\ \mathtt{c})\cdot\mathtt{X}}\ (\approx_\alpha \mathbf{X})$$

Next, we extend the action of a-substitutions to terms and introduce the syntax and semantics of variable substitution for extended terms, axiomatising its application to terms.

2.3 Atom and Variable Substitution

Application of a-substitutions to terms must be defined in a capture-avoidance manner to ensure semantics are preserved under instantiation of variables by terms. Similarly to λ-terms, replacement of variables by terms in extended nominal terms may lead to erroneous binding scopes when a-substitutions act on abstractions.

In order to avoid capture of unabstracted atoms, extended nominal terms rely on freshness contexts and swappings. For any abstraction $[a]s$, there exists a nominal set such that $[a]s \triangleq \{(b, (b\ a)\cdot s) \mid b \in \mathscr{A} \wedge (b = a \vee b\#s)\}$. This is known as the *equivalence class* for the pair (a, s) [Lemma 5.1,(Gabbay and Pitts, 2002)] and sometimes denoted as $[[a]s]_\alpha$. Intuitively, it means $[a]s$ is used as a *representative* of the class, yet there exists an infinite number of terms satisfying such conditions. This notion of the class representative is made explicit when a-substitutions, ϕ, are applied to terms of form $([a]s)$. Then, to avoid capturing any unabstracted atom a which may occur on any of the terms in $\mathscr{I}mg(\phi)$, an α-equivalent term $([b](a\ b)\cdot s)$ from its equivalence class $[[a]s]_\alpha$ is chosen, where $b \in \mathscr{A} \wedge b\#(\{s\} \cup \mathscr{I}mg(\phi))$. Since terms in the image of ϕ may also contain variable symbols, a freshness context $\nabla \subset \{b\#X \mid b \in \mathscr{A} \wedge b \notin A(s, \mathscr{I}mg(\phi)) \wedge X \in V(s, \mathscr{I}mg(\phi))\}$ is therefore necessary to entail the derivation of $[a]s \approx_\alpha [b](a\ b)s$. In the sequel, we say that a freshness context Δ^ϕ *is fresh for* some term t when $\Delta^\phi \subset \{a\#X \mid a \in \mathscr{A} \wedge a \notin A(t) \wedge X \in V(t)\}$.

The action of a-substitutions on terms relies on a freshness context, therefore, we provide below a definition of a term and its related freshness context.

Definition 2.3.1 (Term-in-context)**.** A **term-in-context**, $\nabla \vdash s$, is a pair of a freshness context ∇ and an extended nominal term s. If $\varnothing \vdash s$, we may omit the freshness context and write $\vdash s$ instead.

2.3.1 Atom substitution action

Definition 2.3.2. The **action of a-substitutions ϕ on a term-in-context** $\nabla \vdash t$, written $\nabla \vdash t\phi$, is inductively defined as follows:

$$\nabla \vdash t\,\mathtt{Id} \approx_\alpha t \qquad \nabla \vdash a\phi \approx_\alpha \phi(a) \qquad \nabla \vdash (\phi\hat{}\pi{\cdot}X)\phi' \approx_\alpha (\phi \bullet \phi')\hat{}\pi{\cdot}X$$
$$\nabla \vdash ([a]t)\phi \approx_\alpha [b]((a\ b){\cdot}t)\phi^{-b} \ \text{ (where } \nabla \vdash b\#t, \mathscr{I}mg(\phi))$$
$$\nabla \vdash (ft)\phi \approx_\alpha f(t\phi) \quad \nabla \vdash (t_1,\ldots,t_n)\phi \approx_\alpha (t_1\phi,\ldots,t_n\phi).$$

As explained at the beginning of the section, capture-avoidance a-substitution is guaranteed by choosing a different α-equivalent representative of an abstraction term $[a]t$ with respect to some given freshness context ∇ prior to some a-substitution ϕ acting on such term. There exists an infinite number of atoms which do not appear on either $[a]t$ or ϕ and, since variables have finite support (Pitts, 2003), one can augment ∇ with primitive constraints $b\#X$ for all $X \in (V([a]t) \cup V(\mathscr{I}mg(\phi)))$ and for some $b \in (\mathscr{A} \setminus (A([a]t) \cup A(\mathscr{I}mg(\phi))))$ whenever required. In practice, one could start with a large enough set of new atoms not appearing in the system under consideration and then generate a set of primitive constraints built from such atoms with respect to each variable in the system; such a set would then be combined with the given freshness context before interacting with the system. This approach is similar to the approach taken in (Fairweather et al., 2015; Fernández and Gabbay, 2007) and tacitly assumed in the rest of this paper, right after the following example.

Example 2.3.3 (A-substitution application)**.** The action of a-substitutions $[\mathtt{b} \mapsto \mathtt{c}; \mathtt{c} \mapsto \mathtt{Y}]$ on term-in-context $\vdash [\mathtt{a}]\mathtt{f}(\mathtt{a},\mathtt{b},\mathtt{X})$, using Definition 2.3.2 and augmented with freshness context $\{\mathtt{d\#X}, \mathtt{d\#Y}, \mathtt{e\#X}, \mathtt{e\#Y}\}$, is as follows.

$$\mathtt{d\#X}, \mathtt{d\#Y}, \mathtt{e\#X}, \mathtt{e\#Y} \vdash ([\mathtt{a}]\mathtt{f}(\mathtt{a},\mathtt{b},\mathtt{X}))[\mathtt{b} \mapsto \mathtt{c}; \mathtt{c} \mapsto \mathtt{Y}] \approx_\alpha [\mathtt{d}]\mathtt{f}(\mathtt{d},\mathtt{c},[\mathtt{b} \mapsto \mathtt{c}; \mathtt{c} \mapsto \mathtt{Y}]\hat{}(\mathtt{a}\ \mathtt{d}){\cdot}\mathtt{X}).$$

An alternative composition of a-substitution follows, where both sets of a-substitutions are merged into a single set of mappings.

Lemma 2.3.4 (A-substitution composition). *Let $\phi_1 = [a_1 \mapsto s_1; \ldots; a_n \mapsto s_n], \phi_2$ be a pair of a-substitutions, ∇ a freshness context. Then, given a term s we have,*

$$\nabla \vdash s(\phi_1 \bullet \phi_2) \approx_\alpha s(\phi_2^{-a_1,\ldots,a_n}[a_1 \mapsto s_1\phi_2; \ldots; a_n \mapsto s_n\phi_2])$$

Proof. By induction on the structure of s.

It follows by Definition 2.1.5 where we stated that $s(\phi_1 \bullet \phi_2) = (s\phi_1)\phi_2$, by Definition 2.2.5 for logical equivalence of extended terms and the action of a-substitutions on a term-in-context given in Definition 2.3.2. We omit the inductive proof. ■

The behaviour of our definition of a-substitution corresponds with the notion of higher-order substitution (for CRSs) as we show next.

Lemma 2.3.5. *Let t, s be CRS terms (and therefore also ground nominal terms (see (Domínguez and Fernández, 2014, Property 3.10))). Then, $t[a \mapsto s] \approx_\alpha t\{a \mapsto s\}$ where $t\{a \mapsto s\}$ denotes the term obtained by substituting (using the capture-avoiding substitution of the CRS) a by s in t.*

Proof. By induction on the length of term t. See Appendix A ■

The action of a-substitutions as in Definition 2.3.2 defines the behaviour of explicit substitution rules given in Definition 1.5.4. This theorem is instrumental in Chapter 7 when extending the translation between formalisms to include a-substitutions in NRS systems.

Theorem 2.3.6 (Correctness of a-substitution)**.** *Let t, s be a pair of ground nominal terms. Then, $\nabla \vdash t[a \mapsto s] \approx_\alpha nf_\sigma(\mathtt{sub}([a]t, s))$ where $nf_\sigma(\mathtt{sub}([a]t, s))$ is the normal form of term-in-context $\mathtt{sub}([a]t, s)$ as given in Definition 1.5.11.*

Proof. The theorem follows by (Fernández et al., 2004, Lemma 6.4), Lemma 2.3.5 and the transitive property of α-equivalence for standard nominal terms. ■

2.3.2 Variable substitution action

Substitution of variables by terms is a syntactic meta-level transformation on the structure of extended nominal terms.

Definition 2.3.7. Variable substitutions σ, or just **v-substitutions**, are generated by the grammar: $\sigma ::= \mathtt{Id} \mid [X \mapsto s]\sigma$ where $\mathtt{Id}$ is commonly omitted. Similarly to a-substitutions, v-substitution lists are interpreted as a set of simultaneous mappings, in this case acting on variables in a syntactic manner, without avoiding capture of atoms.

The image of a variable symbol X under v-substitution σ is depicted as $\sigma(X)$ and known as an **instantiation** of X by σ. Write **domain of** σ, $\mathscr{D}om(\sigma)$, for the set of variables such that $\sigma(X) \neq X$ for all $X \in \mathscr{X}$. Call **image of** σ, written $\mathscr{I}mg(\sigma)$, the set of terms $\{\sigma(X) \mid X \in \mathscr{D}om(\sigma)\}$. The **restriction of a substitution** σ to a set of variables V, written $\sigma|_V$, is defined as $\sigma|_V = [X \mapsto \sigma(X) \mid X \in V]$.

In the sequel, a sequential list of simultaneous v-substitution mappings $[X_1 \mapsto s_1] \cdots [X_n \mapsto s_n]$ is represented as $[X_1 \mapsto s_1; \ldots; X_n \mapsto s_n]$.

We continue by extending the action of v-substitutions to terms.

Application of v-substitutions, σ, to variable occurrences, $\phi\hat{}\pi{\cdot}X$, induces the action of a-substitutions on instantiations of X by σ, $\sigma(X)$. Accordingly, the action of v-substitutions on terms is also parametrised by freshness contexts. However, it is left implicit on the definitions below.

Definition 2.3.8. The **action of v-substitutions σ on a-substitutions** is defined as

$$([a_1 \mapsto s_1; \ldots; a_n \mapsto s_n])\sigma \triangleq [a_1 \mapsto s_1\sigma; \ldots; a_n \mapsto s_n\sigma]$$

where $s_i\sigma$ is the application of σ to term s_i as defined in Definition 2.3.9, for $1 \leq i \leq n$.

Definition 2.3.9. The **action of v-substitutions σ on a term** t, written $t\sigma$, is defined by induction on terms as follows:

$$t\,\mathtt{Id} \triangleq t \qquad a\sigma \triangleq a \qquad (\phi\hat{}\pi{\cdot}X)\sigma \triangleq (\pi{\cdot}\sigma(X))(\phi\sigma)$$
$$(\phi\hat{}\pi{\cdot}Y)\sigma \triangleq (\phi\sigma)\hat{}\pi{\cdot}Y \;\; (X \neq Y) \qquad ([a]t)\sigma \triangleq [a](t\sigma)$$
$$(ft)\sigma \triangleq f(t\sigma) \qquad (t_1, \ldots, t_n)\sigma \triangleq (t_1\sigma, \ldots, t_n\sigma).$$

Call a term t an **instance** of a term s when t is the result of applying σ to s, $s\sigma$.

Composition of variable substitutions, written $\sigma \bullet \sigma'$, is applied inversely since the notation is postfix. Therefore $s(\sigma \bullet \sigma') = (s\sigma)\sigma'$.

Sometimes, an alternative version of v-substitution composition is preferred when establishing some proofs. Such version is denoted below.

Lemma 2.3.10 (Composition of v-substitutions)**.** *Let θ, σ be any two v-substitutions, t a nominal term and ∇ a freshness context. Then,*

$$\nabla \vdash t(\sigma \bullet \theta) \approx_\alpha t(\theta|_V \sigma')$$

where $V = (\mathscr{D}om(\theta) \setminus \mathscr{D}om(\sigma))$ and $\sigma' = [X \mapsto (\sigma(X))\theta \mid X \in \mathscr{D}om(\sigma)]$.

Proof. By induction on the structure of t.

It follows by Definition 2.3.7 where we stated that $t(\sigma \bullet \theta) = (t\sigma)\theta$, by Definition 2.2.5 for logical equivalence of extended terms and the action of v-substitutions on terms given in Definition 2.3.9. We omit the inductive proof. ■

Example 2.3.11 (V-substitution application)**.** The action of v-substitution $[\mathtt{X} \mapsto \mathtt{f}(\mathtt{a}, [\mathtt{b}]\mathtt{Y}); \mathtt{Y} \mapsto \mathtt{b}]$ on term $[\mathtt{a}]\mathtt{g}([\mathtt{a} \mapsto \mathtt{Y}; \mathtt{b} \mapsto \mathtt{a}] \cdot \mathtt{X})$, using Definition 2.3.9, is as follows.

$$\mathtt{c\#Y} \vdash [\mathtt{a}]\mathtt{g}(\mathtt{f}(\mathtt{b}, [\mathtt{c}][\mathtt{a} \mapsto \mathtt{b}] \cdot (\mathtt{a}\ \mathtt{b}) \cdot \mathtt{Y})).$$

Above, note how the freshness context has been augmented with primitive constraint $\mathtt{c\#Y}$ in order to deal with the application of the a-substitution after instantiating variable $\mathtt{X}$ so that an α-equivalent abstraction term could be chosen to avoid any potential variable capture. Observe also that the mapping with atom $\mathtt{b}$ in the domain of the suspended a-substitution on variable $\mathtt{X}$ was discarded when applied to abstractor $[\mathtt{b}]-$, following Definition 2.3.2.

Sometimes we need to represent the **pointwise application of permutations to v-substitutions**, the notation being $(\pi \cdot \sigma) \triangleq \{\pi \cdot \sigma(X) \mid X \in \mathscr{D}om(\sigma)\}$. For instance, $(a\ b) \cdot ([X \mapsto a; Y \mapsto [a]Z]) = [X \mapsto b; Y \mapsto [b](a\ b) \cdot Z]$.

The following notation regarding v-substitutions will be of use later.

Definition 2.3.12 (v-substitution notation)**.**

- Let σ, σ' be a pair of v-substitutions and ∇ a freshness context. We write $\nabla \vdash \sigma \approx_\alpha \sigma'$ as an abbreviation of $\nabla \vdash \sigma(X) \approx_\alpha \sigma'(X)$ for all $X \in \mathscr{D}om(\sigma) \cup \mathscr{D}om(\sigma')$.

- If C is some freshness constraint $a\#s$ or α-equality constraint $s \approx_\alpha t$, write: $C\sigma$ for $a\#s\sigma$ or $s\sigma \approx_\alpha t\sigma$ respectively. Moreover, if $C = \bot$ (resp. $C = \top$), then $\bot\sigma = \bot$ (resp. $\top\sigma = \top$). Also, $\varnothing\sigma = \varnothing$.

2.4 Alternative Inference Rules

Next, an alternative set of inference rules for both freshness and α-equality relations is introduced. The distinction with respect to Definition 2.2.3 and Definition 2.2.5 is that the premise of both core variable rules, that is, rule (#**X**) and rule ($\approx_\alpha$**X**), is now written using an auxiliary function returning the set of atoms which are to be fresh with respect to the given variable symbol for the relation to hold. This approach is more convenient for the description of theorems, properties and their respective proofs since freshness contexts are given as input, whereas Definition 2.2.3 and Definition 2.2.5 are more suitable for an algorithmic implementation as we show in Chapter 3. The pair of alternative inference rules along with their proof of equivalence with respect to the rules given in Section 2.2 is introduced below.

2.4.1 Alternative freshness rules

Definition 2.4.1 (Alternative freshness rules)**.** Call **(alternative) freshness rules** the set of axioms and inference rules from Definition 2.2.3 where core rule (#**X**) has been replaced by rule (#$_{\mathbf{alt}}$**X**) given below. Write $\nabla \vdash_{alt} a\#t$ for the freshness judgement derived using the alternative rules.

$$\frac{\forall b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi) : b\#X \in \nabla}{\nabla \vdash_{alt} a\#\phi\hat{}\pi\cdot X} \ (\#_{\mathbf{alt}}\mathbf{X})$$

The notation $\forall b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi) : b\#X \in \nabla$ in the premise of rule (#$_{\mathbf{alt}}$**X**) is an auxiliary function which ensures that a particular atom a is not introduced as an unabstracted atom for any instance of $\phi\hat{}\pi\cdot X$ with respect to some freshness context ∇, with notation $\mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ being an abbreviation for the following finite set of atoms:

$$\mathtt{fresh}(\nabla, a, \phi\hat{}\pi) \triangleq \{b \mid b \in \mathscr{A} \wedge \nabla \nvdash_{alt} a\#\phi(\pi(b))\}. \tag{2.1}$$

More succinctly,

$$\mathtt{fresh}(\nabla, a, \phi\hat{}\pi) \triangleq \{b \mid \pi(b) \in (\mathscr{D}om(\phi) \cup \{a\}) \wedge \nabla \nvdash_{alt} a\#\phi(\pi(b))\}. \tag{2.2}$$

Informally, $\mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ is the least set of atoms which must be fresh for X in ∇ in order to entail the derivation of judgement $\nabla \vdash_{alt} a\#\phi\hat{}\pi\cdot X$. Observe that $\forall b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi) : b\#X \in \nabla$ is not a negated premise but the left-hand side of an implication, with $\nabla \nvdash_{alt} a\#\phi(\pi(b))$ being on the right-hand side.

Conditions for the construction of set $\mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ can be seen as a terminating recursive call on the function arguments for each atom $\pi(b) \in (\mathscr{D}om(\phi) \cup \{a\})$ (see Equation 2.2) and finite mappings ϕ. Also, $\forall b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi) : b\#X \in \nabla$ is both syntax-directed and structurally of smaller size than its conclusion since a-substitutions occur as subterms in the syntax of variables. An example follows.

Example 2.4.2. Freshness constraint $\mathtt{a}\#[\mathtt{b} \mapsto \mathtt{Y}]\hat{}(\mathtt{a}\ \mathtt{c})\cdot\mathtt{X}$ from Example 2.2.4 is derivable from $\nabla_1 = \{\mathtt{a}\#\mathtt{Y}, \mathtt{c}\#\mathtt{X}\}$ or from $\nabla_2 = \{\mathtt{b}\#\mathtt{X}, \mathtt{c}\#\mathtt{X}\}$, using rule (#$_{\mathbf{alt}}$**X**) from Definition 2.4.1.

Let $[\mathtt{b} \mapsto \mathtt{Y}] = \phi$ and $(\mathtt{a}\ \mathtt{c}) = \pi$.

- From context ∇_1 we compute $\mathtt{fresh}(\nabla_1, \mathtt{a}, \phi\hat{}\pi) = \{\mathtt{c}\}$ as follows:
  $\mathscr{D}\mathtt{om}(\phi) \cup \{\mathtt{a}\} = \{\mathtt{a}, \mathtt{b}\}$ and there are 2 cases to check:

  - (Case $\pi(\mathtt{c}) \in (\mathscr{D}\mathtt{om}(\phi) \cup \{\mathtt{a}\})$).

Since $\pi(\mathtt{c}) = \mathtt{a}$ and $\phi(\pi(\mathtt{c})) = \mathtt{a}$, it follows that $\nabla_1 \not\vdash_{alt} \mathtt{a}\#\phi(\pi(\mathtt{c}))$. Therefore $\mathtt{c} \in \mathtt{fresh}(\nabla_1, \mathtt{a}, \phi \hat{} \pi)$.

– (Case $\pi(\mathtt{b}) \in (\mathscr{D}\mathtt{om}(\phi) \cup \{\mathtt{a}\})$).
Since $\pi(\mathtt{b}) = \mathtt{b}$ and $\phi(\pi(\mathtt{b})) = \mathtt{Y}$, it follows that $\dfrac{\mathtt{a}\#\mathtt{Y} \in \nabla_1}{\nabla_1 \vdash_{alt} \mathtt{a}\#\phi(\pi(\mathtt{b}))}\ (\#_{\mathbf{alt}}\mathbf{X})$.
Therefore $\mathtt{b} \notin \mathtt{fresh}(\nabla_1, \mathtt{a}, \phi \hat{} \pi)$.

Hence,
$$\frac{\mathtt{c}\#\mathtt{X} \in \nabla_1}{\nabla_1 \vdash_{alt} \mathtt{a}\#[\mathtt{b} \mapsto \mathtt{Y}] \hat{} (\mathtt{a}\ \mathtt{c}) \cdot \mathtt{X}}\ (\#_{\mathbf{alt}}\mathbf{X})$$

- From context ∇_2 we compute $\mathtt{fresh}(\nabla_2, \mathtt{a}, [\mathtt{b} \mapsto \mathtt{Y}] \hat{} (\mathtt{a}\ \mathtt{c})) = \{\mathtt{b}, \mathtt{c}\}$ as follows: $\mathscr{D}\mathtt{om}(\phi) \cup \{\mathtt{a}\} = \{\mathtt{a}, \mathtt{b}\}$ and there are 2 cases to check:

  – (Case $\pi(\mathtt{c}) \in (\mathscr{D}\mathtt{om}(\phi_1) \cup \{\mathtt{a}\})$).
  Since $\pi(\mathtt{c}) = \mathtt{a}$ and $\phi(\pi(\mathtt{c})) = \mathtt{a}$, it follows that $\nabla_2 \not\vdash_{alt} \mathtt{a}\#\phi(\pi(\mathtt{c}))$.
  Therefore $\mathtt{c} \in \mathtt{fresh}(\nabla_2, \mathtt{a}, \phi \hat{} \pi)$.

  – (Case $\pi(\mathtt{b}) \in (\mathscr{D}\mathtt{om}(\phi) \cup \{\mathtt{a}\})$).
  Since $\pi(\mathtt{b}) = \mathtt{b}$, $\phi(\pi(\mathtt{b})) = \mathtt{Y}$ and $\mathtt{a}\#\mathtt{Y} \notin \nabla_2$, it follows that $\nabla_2 \not\vdash_{alt} \mathtt{a}\#\phi(\pi(\mathtt{b}))$.
  Therefore $\mathtt{b} \in \mathtt{fresh}(\nabla_2, \mathtt{a}, \phi \hat{} \pi)$.

Hence,
$$\frac{\mathtt{b}, \mathtt{c}\#\mathtt{X} \in \nabla_2}{\nabla_2 \vdash_{alt} \mathtt{a}\#[\mathtt{b} \mapsto \mathtt{Y}] \hat{} (\mathtt{a}\ \mathtt{c}) \cdot \mathtt{X}}\ (\#_{\mathbf{alt}}\mathbf{X})$$

Below, we demonstrate core and alternative freshness inference rules are indeed equivalent.

Lemma 2.4.3. *Let ∇ be a freshness context, $a\#t$ a freshness relation between any atom a and term t. Then, $\nabla \vdash a\#t \iff \nabla \vdash_{alt} a\#t$, using the derivation rules from Definitions 2.2.3 & 2.4.1 respectively.*

Proof. By induction on the structure of t. See Appendix A. ■

In the sequel, we make no distinction between judgements derived with either system of rules, denoting both by symbol $\vdash$. Moreover, we tacitly assumed the application of alternative freshness inference rules in this and future chapters unless stated otherwise.

2.4.2 Alternative α-equality rules

Similar to rule ($\#_{\mathbf{alt}}\mathbf{X}$), we offer an alternate interpretation of rule ($\approx_\alpha\mathbf{X}$) where the notion of disagreement set is introduced in the premise of the rule.

Definition 2.4.4 (Alternative α-equivalence rules)**.** Call **(Alternative) α-equivalence rules** the set of axioms and inference rules from Definition 2.2.5 where rule ($\approx_\alpha\mathbf{X}$) has been replaced by rule ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{X}$) given below. Write $\nabla \vdash_{alt} s \approx_\alpha t$ for the α-equality judgement derived using the core rules.

$$\frac{\forall a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi') : a\#X \in \nabla}{\nabla \vdash_{alt} \phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot X} \ (\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$$

The notation to build the difference set, written $\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$, extends previous work by finding the set of atoms which, under application of atom actions from both sides of the equation, do not resolve to logically equal terms. More formally,

$$\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi') \triangleq \{a \mid a \in \mathscr{A} \wedge \nabla \nvdash_{alt} \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))\} \tag{2.3}$$

which can be represented more concisely by the equivalent set

$$\{a \mid a \in (\mathscr{D}omP(\phi, \phi') \cup \mathscr{S}upportP(\pi, \pi')) \wedge \nabla \nvdash_{alt} \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))\}. \tag{2.4}$$

Similar to the notation in the premise of rule ($\#_{\mathbf{alt}}\mathbf{X}$), a freshness context is required during the computation of function ds due to the possible occurrence of variables on the terms in the image of a-substitutions. Also, observe that $\forall a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi') : a\#X \in \nabla$ is not a negated premise but the left-hand side of an implication, with $\nabla \nvdash_{alt} \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))$ on its right-hand side.

The conditions for the formation of set $\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ for any pair of atom actions $\phi\hat{}\pi, \phi'\hat{}\pi'$ and freshness context ∇ can be seen as a terminating recursive call on the function arguments since a-substitutions are finite mappings. Moreover, terms in the image of a-substitutions occur as subterms suspended on the syntax of variables, leading to the conclusion that the premise of rule ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{X}$) is both syntax-directed and structurally of smaller size than the conclusion. Below we offer an example of application.

Example 2.4.5. The α-equality constraint $[\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b} \mapsto \mathtt{Y}]\cdot(\mathtt{b}\ \mathtt{c})\cdot\mathtt{X}$ from Example 2.2.6 is derivable from $\nabla_1 = \{\mathtt{b\#X}, \mathtt{a\#Y}, \mathtt{b\#Y}\}$ or from $\nabla_2 = \{\mathtt{b\#X}, \mathtt{c\#X}\}$, using rule $(\approx\alpha_{\mathbf{alt}}\mathbf{X})$ from Definition 2.4.4.

Let $[\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}] = \phi_1$ and $\mathtt{Id} = \pi_1$. Also, let $[\mathtt{b} \mapsto \mathtt{Y}] = \phi_2$ and $(\mathtt{b}\ \mathtt{c}) = \pi_2$.

- From context ∇_1 we compute $\mathtt{ds}(\nabla_1, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2) = \{\mathtt{b}\}$ as follows:
  $\mathscr{D}\mathtt{omP}(\phi_1, \phi_2) \cup \mathscr{S}\mathtt{upportP}(\pi_1, \pi_2) = \{\mathtt{b}, \mathtt{c}\}$ and there are 2 cases to check:

  - (Case $\mathtt{b} \in (\mathscr{D}\mathtt{omP}(\phi_1, \phi_2) \cup \mathscr{S}\mathtt{upportP}(\pi_1, \pi_2))$).
    Since $\phi_1(\pi_1(\mathtt{b})) = \mathtt{b}$ and $\phi_2(\pi_2(\mathtt{b})) = \mathtt{c}$,
    it follows that $\nabla_1 \vdash_{alt} \phi_1(\pi_1(\mathtt{b})) \not\approx_\alpha \phi_2(\pi_2(\mathtt{b}))$.
    Therefore $\mathtt{b} \in \mathtt{ds}(\nabla_1, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$.

  - (Case $\mathtt{c} \in (\mathscr{D}\mathtt{omP}(\phi_1, \phi_2) \cup \mathscr{S}\mathtt{upportP}(\pi_1, \pi_2))$).
    Since $\phi_1(\pi_1(\mathtt{c})) = (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}$ and $\phi_2(\pi_2(\mathtt{c})) = \mathtt{Y}$,
    it follows that $\dfrac{\mathtt{a}, \mathtt{b\#Y} \in \nabla_1}{\nabla_1 \vdash_{alt} \phi_1(\pi_1(\mathtt{c})) \approx_\alpha \phi_2(\pi_2(\mathtt{c}))}\ (\approx\alpha_{\mathbf{alt}}\mathbf{X})$.
    Therefore $\mathtt{c} \notin \mathtt{ds}(\nabla_1, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$.

  Hence,
  $$\frac{\mathtt{b\#X} \in \nabla_1}{\nabla_1 \vdash_{alt} [\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b} \mapsto \mathtt{Y}]\cdot(\mathtt{b}\ \mathtt{c})\cdot\mathtt{X}}\ (\approx\alpha_{\mathbf{alt}}\mathbf{X})$$

- From context ∇_2 we compute $\mathtt{ds}(\nabla_2, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2) = \{\mathtt{b}, \mathtt{c}\}$ as follows:
  $\mathscr{D}\mathtt{omP}(\phi_1, \phi_2) \cup \mathscr{S}\mathtt{upport}(\pi_1)\pi_2 = \{\mathtt{b}, \mathtt{c}\}$ and there are 2 cases to check:

  - (Case $\mathtt{b} \in (\mathscr{D}\mathtt{omP}(\phi_1, \phi_2) \cup \mathscr{S}\mathtt{upportP}(\pi_1, \pi_2))$).
    Since $\phi_1(\pi_1(\mathtt{b})) = \mathtt{b}$ and $\phi_2(\pi_2(\mathtt{b})) = \mathtt{c}$,
    it follows that $\nabla_1 \vdash_{alt} \phi_1(\pi_1(\mathtt{b})) \not\approx_\alpha \phi_2(\pi_2(\mathtt{b}))$. Therefore $\mathtt{b} \in \mathtt{ds}(\nabla_1, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$.

  - (Case $\mathtt{c} \in (\mathscr{D}\mathtt{omP}(\phi_1, \phi_2) \cup \mathscr{S}\mathtt{upportP}(\pi_1, \pi_2))$).
    Since $\phi_1(\pi_1(\mathtt{c})) = (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}$, $\phi_2(\pi_2(\mathtt{c})) = \mathtt{Y}$ and $\mathtt{a}, \mathtt{b\#Y} \notin \nabla_2$,
    it follows that $\nabla_2 \vdash_{alt} \phi_1(\pi_1(\mathtt{c})) \not\approx_\alpha \phi_2(\pi_2(\mathtt{c}))$.
    Therefore $\mathtt{c} \in \mathtt{ds}(\nabla_2, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$.

Hence,

$$\frac{\mathtt{b},\mathtt{c}\#\mathtt{X} \in \nabla_2}{\nabla_2 \vdash_{alt} [\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b} \mapsto \mathtt{Y}]\cdot(\mathtt{b}\ \mathtt{c})\cdot\mathtt{X}}\ (\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$$

The following lemma shows that both core and alternative inference rules are equivalent.

Lemma 2.4.6. *Let ∇ be a freshness context, $s \approx_\alpha t$ an α-equality relation between any pairs of terms s,t. Then, $\nabla \vdash s \approx_\alpha t \iff \nabla \vdash_{alt} s \approx_\alpha t$, using the derivation rules from Definitions 2.2.5 & 2.4.4, respectively.*

Proof. By induction on the derivation of $s \approx_\alpha t$. The proof is similar to that in Lemma 2.4.3 and therefore omitted. ■

In the sequel, we do not make a distinction between judgements derived with either system of rules, denoting both by symbol $\vdash$. Moreover, we tacitly assume the application of alternative α-equality inference rules in this and future chapters unless stated otherwise.

In the next section, we prove some important properties about the freshness and α-equality relation.

2.5 Properties of Freshness and α-equivalence

This section centers on proving expected properties of the extended framework. The main property of this section is stated in Theorem 2.5.8 where we prove that $\approx_\alpha$ is an equivalence relation. A fundamental property for eNRSs is stated in Theorem 2.5.19, claiming that α-equivalence is invariant under application of both atom actions, π and ϕ. We end the section with Corollary 2.5.20, showing α-equivalence to also be a congruence.

When stating the claims, we tacitly assume all judgements to hold. Notice that all constraints are derivable from falsity, $\bot$, hence our effort is on proving properties for the cases where relation $\vdash$ holds.

The section begins by generalising some properties of standard nominal terms to extended terms. They will be applied throughout this section and on further chapters. It continues by stating, in Lemma 2.5.5, that semantics of both freshness and alpha-equality constraints are preserved under name swappings. This notion of invariance up to permuted renaming of atoms is required to support our claim of predicate $\approx_\alpha$ being an equivalence relation (see Theorem 2.5.8).

Below, we prove that entailment of freshness and α-equality constraints by a given freshness context is preserved under extension of the set of primitive constraints.

Lemma 2.5.1. *(Weakening) Let ∇ be a freshness context and s,t any pair of terms. Assume $X \in V(s,t)$ Then,*

- *if* $\nabla \vdash a\#s$, *then* $\{b\#X\} \cup \nabla \vdash a\#s$;
- *if* $\nabla \vdash s \approx_\alpha t$, *then* $\{b\#X\} \cup \nabla \vdash s \approx_\alpha t$.

Proof. By routine induction on each derivation rule from Definition 2.4.1 and Definition 2.4.4. We show only the variable cases.

For the first part, suppose $s = \phi\hat{}\pi{\cdot}X$ and $b \in \mathscr{A}$. Then, $\nabla \vdash a\#\phi\hat{}\pi{\cdot}X$ and we must prove the following cases, the case where $b \notin A(\phi\hat{}\pi{\cdot}X)$ which it follows trivially and the case where $b \in A(\phi\hat{}\pi{\cdot}X)$. For the latter we observe that, by using rule $(\#_{\mathbf{altX}})$, either $\pi(b) \in (\mathscr{D}om(\phi) \cup \{a\})$ or not. For the case where $\pi(b) \notin (\mathscr{D}om(\phi) \cup \{a\})$, it is a fact that $\nabla \vdash a\#\phi(\pi(b))$ and the result follows by the inductive step. Similarly for the case where $\pi(b) \in (\mathscr{D}om(\phi) \cup \{a\}) \wedge \nabla \vdash a\#\phi(\pi(b))$. Lastly, for the case where $\pi(b) \in (\mathscr{D}om(\phi) \cup \{a\}) \wedge \nabla \nvdash a\#\phi(\pi(b))$, then it is the case that $b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ such that $b\#X \in \nabla$ and thus it holds.

The second part is similarly solved and thus omitted here. ■

However, adding elements to the freshness context of a judgement has an effect on the premise of both derivation rules for variables, namely $(\#_{\mathbf{altX}})$ and $(\approx\alpha_{\mathbf{altX}})$, it decreases the size of the generated set of atoms. In other words, the more assumptions added to the set of constraints, the more evidence there is to derive judgements. This is a nice property that we formalise below.

Property 2.5.2.
- *Suppose* Δ *is a freshness context. Suppose also that* $\nabla \vdash \phi\hat{}\pi{\cdot}X \approx_\alpha \phi'\hat{}\pi'{\cdot}X$. *Then, for any* $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$, *if* $\Delta, \nabla \vdash \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))$, *it is the case that* $a \notin \mathtt{ds}(\Delta \cup \nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$.
- *Suppose* Δ *is a freshness context. Suppose also that* $\nabla \vdash n\#\phi\hat{}\pi{\cdot}X$. *Then, for any* $a \in \mathtt{fresh}(\nabla, n, \phi\hat{}\pi)$, *if* $\Delta, \nabla \vdash n\#\phi(\pi(a))$, *it is the case that* $a \notin \mathtt{fresh}(\Delta \cup \nabla, n, \phi\hat{}\pi)$.

Proof. The first case follows directly by the definition of disagreement set (see Equation 2.3) whereas the second case follows directly by the definition of `fresh` (see Equation 2.1). ■

Sometimes we need to *strengthen* a judgement by discarding superfluous primitive constraints from the freshness context. We establish this property in the lemma below.

Lemma 2.5.3 (Strengthening)**.**
- *Assume* $X \in V(s)$. *If* $\{b\#X\} \cup \nabla \vdash a\#s$ *and* $b \notin A(s)$ *then* $\nabla \vdash a\#s$.
- *Assume* $X \in V(s,t)$. *If* $\{b\#X\} \cup \nabla \vdash s \approx_\alpha t$ *and* $b \notin A(s,t)$ *then* $\nabla \vdash s \approx_\alpha t$.

Proof. We transform a derivation of $\{b\#X\} \cup \nabla \vdash a\#s$ (resp. $\{b\#X\} \cup \nabla \vdash s \approx_\alpha t$) into a derivation of $\nabla \vdash a\#s$ (resp. $\nabla \vdash s \approx_\alpha t$).

- For the freshness judgement.

  All cases have been previously proved for standard terms in (Gabbay and Mathijssen, 2008, Lemma 2.22). We extend the lemma to the variable case for extended terms.

  – Suppose $\{b\#X\} \cup \nabla \vdash \mathtt{fresh}(\{b\#X\} \cup \nabla, a, \phi\hat{}\pi)\#X$ where $b \notin \mathtt{fresh}(\{b\#X\} \cup \nabla, a, \phi\hat{}\pi)$ such that $\{b\#X\} \cup \nabla \vdash a\#\phi\hat{}\pi\cdot X$ is derivable using ($\#_{\mathbf{alt}}\mathbf{X}$). Applying the inductive hypothesis to the unpacked definition of $\mathtt{fresh}(\{b\#X\} \cup \nabla, a, \phi\hat{}\pi)$ we obtain $\{n \mid n \in \mathscr{A}, \nabla \nvdash a\#\phi(\pi(n))\}$. The result follows by application of rule ($\#_{\mathbf{alt}}\mathbf{X}$).

- For the α-equality judgement.

  We prove the variable case. The proof for the rest of the cases is straightforward and thus omitted.

  – Suppose $\{b\#X\} \cup \nabla \vdash \mathtt{ds}(\{b\#X\} \cup \nabla, \phi\hat{}\pi, \phi'\hat{}\pi')\#X$ where $b \notin \mathtt{ds}(\{b\#X\} \cup \nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ such that $\{b\#X\} \cup \nabla \vdash \phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot X$ is derivable using ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$). Applying the inductive hypothesis to the unpacked definition of $\mathtt{ds}(\{b\#X\} \cup \nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ we obtain $\{a \mid a \in \mathscr{A}, \nabla \nvdash \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))\}$. The result follows by application of rule ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$).

■

In Section 2.3.1, it was shown that some judgements are weakened when a-substitutions act on abstraction terms $([a]t)$ (although the notation was left implicit). By extending the set of primitive constraints additional evidence is provided to find another representative from the equivalence class of pair (a,t) and thus avoid capturing atoms under erroneous binding scopes. The property of strengthening is useful to recover the least set of primitive constraints that entails the derivation proof of a freshness or α-equality relation. Take for instance Example 2.3.3, using the strengthening property we obtain $\mathtt{d\#X}, \mathtt{d\#Y} \vdash ([\mathtt{a}]\mathtt{f}(\mathtt{a},\mathtt{b},\mathtt{X}))[\mathtt{b} \mapsto \mathtt{c}; \mathtt{c} \mapsto \mathtt{Y}] \approx_\alpha [\mathtt{d}]\mathtt{f}(\mathtt{d},\mathtt{c},[\mathtt{b} \mapsto \mathtt{c}; \mathtt{c} \mapsto \mathtt{Y}]\hat{}(\mathtt{a}\ \mathtt{d})\cdot\mathtt{X})$ where subset $\{\mathtt{e\#X}, \mathtt{e\#Y}\}$ has been discarded from the freshness context.

The following lemma extends some well-known properties of standard nominal terms to our extended framework.

Lemma 2.5.4. *Fix a freshness context* ∇.

1. $\nabla \vdash \pi\cdot s \approx_\alpha t \iff \nabla \vdash s \approx_\alpha \pi^{-1}\cdot t$;
2. $\nabla \vdash s \approx_\alpha (\pi^{-1} \circ \pi)\cdot t \iff \nabla \vdash s \approx_\alpha t$;
3. $\nabla \vdash a\#\pi\cdot s \iff \nabla \vdash \pi^{-1}(a)\#s$;

4. $\nabla \vdash a\#X \iff a\#X \in \nabla$.

Proof. Proofs for claim 1 and claim 3 can be found in (Fernández and Gabbay, 2007, Lemma 20), whereas claim 2 is in (Urban et al., 2004, Corollary 2.12). We update the case for variables.

1. Both directions are proved similarly, therefore we prove just the right implication. The proof is done by induction on the derivation of $\nabla \vdash \pi{\cdot}s \approx_\alpha t$. The case for rule $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$ goes as follows. Suppose $a\#X \in \nabla$ for all

   $a \in \mathtt{ds}(\nabla, (\pi{\cdot}\phi_s)\hat{}(\pi \circ \pi_s), \phi_t\hat{}\pi_t)$ such that $\nabla \vdash (\pi{\cdot}\phi_s)\hat{}(\pi \circ \pi_s){\cdot}X \approx_\alpha \phi_t\hat{}\pi_t{\cdot}X$ is derivable using $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$. By application of Definition 2.1.8 on the LHS we obtain $\nabla \vdash \pi{\cdot}(\phi_s\hat{}\pi_s{\cdot}X) \approx_\alpha \phi_t\hat{}\pi_t{\cdot}X$. Applying the inductive step to the unpacked definition of $\mathtt{ds}(\nabla, (\pi{\cdot}\phi_s)\hat{}(\pi \circ \pi_s), \phi_t\hat{}\pi_t)$ we obtain $\{a \mid a \in \mathscr{A} \wedge \nabla \vdash \phi_s(\pi_s(a)) \approx_\alpha \pi^{-1}{\cdot}(\phi_t(\pi_t(a)))\}$. The result then follows by rule $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$.

2. Both directions are also proved similarly, therefore we prove just the right implication. The proof is done by induction on the derivation of $\nabla \vdash s \approx_\alpha (\pi^{-1} \circ \pi){\cdot}t$. The case for rule $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$ goes as follows. Suppose $a\#X \in \nabla$ for all

   $a \in \mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, ((\pi^{-1} \circ \pi){\cdot}\phi_t)\hat{}((\pi^{-1} \circ \pi){\cdot}\pi_t))$ such that

   $\nabla \vdash \phi_s\hat{}\pi_s{\cdot}X \approx_\alpha ((\pi^{-1} \circ \pi){\cdot}\phi_t)\hat{}((\pi^{-1} \circ \pi){\cdot}\pi_t)$ is derivable using $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$. By application of Definition 2.1.8 on the LHS we obtain, $\nabla \vdash \phi_s\hat{}\pi_s{\cdot}X \approx_\alpha (\pi^{-1} \circ \pi){\cdot}(\phi_t\hat{}\pi_t{\cdot}X)$.

   Applying the inductive step to the unpacked definition of

   $\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, ((\pi^{-1} \circ \pi){\cdot}\phi_t)\hat{}((\pi^{-1} \circ \pi){\cdot}\pi_t))$ we obtain $\{a \mid a \in \mathscr{A} \wedge \nabla \vdash s \approx_\alpha t\}$. Then, the result follows by rule $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$.

3. Both directions are also proved similarly, therefore we prove just the right implication. The proof is done by induction on the derivation of $\nabla \vdash a\#\pi{\cdot}s$. For the case of rule $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$, suppose $n\#X \in \nabla$ for all $n \in \mathtt{fresh}(\nabla, a, (\pi{\cdot}\phi_s)\hat{}(\pi{\cdot}\pi_s))$ such that $\nabla \vdash a\#(\pi{\cdot}\phi_s)\hat{}(\pi{\cdot}\pi_s){\cdot}X$ is derivable using $(\#_{\mathbf{alt}}\mathbf{X})$. By application of Definition 2.1.8 we obtain, $\nabla \vdash a\#\pi{\cdot}(\phi_s\hat{}\pi_s{\cdot})$. By application of the inductive step to the unpacked definition of $\mathtt{fresh}(\nabla, a, (\pi{\cdot}\phi_s)\hat{}(\pi{\cdot}\pi_s))$ we obtain $\{n \mid n \in \mathscr{A} \wedge \nabla \vdash \pi^{-1}(a)\#\phi_s(\pi_s(n))\}$. Then, the result follows by rule $(\#_{\mathbf{alt}}\mathbf{X})$.

The proof of claim 4 follows from the syntax-directed nature of the freshness rules given in Definition 2.4.1 and the particular case of rule $(\#_{\mathbf{alt}}\mathbf{X})$ where atom actions are `Id`. ■

Freshness and α-equality judgements are preserved under name swapping action.

Lemma 2.5.5.

1. $\nabla \vdash a\#s$ *if and only if* $\nabla \vdash \pi\cdot a\#\pi\cdot s$.

2. $\nabla \vdash s \approx_\alpha t$ *if and only if* $\nabla \vdash \pi\cdot s \approx_\alpha \pi\cdot t$.

Proof. The first part is by induction on the derivation of $\nabla \vdash a\#s$. The second part is by induction on the derivation of $\nabla \vdash s \approx_\alpha t$. Both proofs can be found in (Fernández and Gabbay, 2007, Lemma 20). We update the case for extended variables.

For claim 1.

The *only if* direction.

- The case ($\#_{\mathbf{alt}}\mathbf{X}$). Suppose $b\#X \in \nabla$ for all $b \in \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$. Then, $\nabla \vdash a\#\phi_1\hat{}\pi_1\cdot X$ is derivable using ($\#_{\mathbf{alt}}\mathbf{X}$). By inductive hypothesis on the unpacked definition of $\mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ we obtain $\{b \mid b \in \mathscr{A} \wedge \nabla \not\vdash \pi\cdot a\#\pi\cdot(\phi_1(\pi_1(b)))\}$. Hence $\mathtt{fresh}(\nabla, \pi(a), (\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1)) = \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ and the result follows by using ($\#_{\mathbf{alt}}\mathbf{X}$) and Definition 2.1.8.

For claim 2.

The *only if* direction.

- The case ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$). Suppose $a\#X \in \nabla$ for all $a \in \mathtt{ds}(\nabla, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$. Then, $\nabla \vdash \phi_1\hat{}\pi_1\cdot X \approx_\alpha \phi_2\hat{}\pi_2\cdot X$ is derivable using ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$). By inductive hypothesis on the unpacked definition of $\mathtt{ds}(\nabla, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$ we obtain $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \pi\cdot(\phi_1(\pi_1(a))) \approx_\alpha \pi\cdot(\phi_2(\pi_2(a)))\}$. Hence $\mathtt{ds}(\nabla, (\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1), (\pi\cdot\phi_2)\hat{}(\pi\circ\pi_2)) = \mathtt{ds}(\nabla, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$ and the result follows by using ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$) and Definition 2.1.8.

The *if* direction for both claims follows by Lemma 2.5.4, claim 1, 2 and 3. We omit both proofs. ■

The following technical lemma will be useful to prove the properties stated in Theorem 2.5.8. It shows that distinct permutations acting on the same term can produce α-equivalent terms if the atoms characterised by the difference set are fresh for the term. An extended version of the lemma, dealing with both atoms actions, is stated later in Lemma 2.5.14. Distinction between the standard and the extended lemma is due to the former providing support for Theorem 2.5.8 whereas the latter needs such theorem for support.

Lemma 2.5.6. *If* $\nabla \vdash a\#s$ *for each* $a \in \mathtt{ds}(\nabla, \pi, \pi')$, *then* $\nabla \vdash \pi\cdot s \approx_\alpha \pi'\cdot s$.

Proof. By induction of the structure of s. The proof can be found in (Fernández and Gabbay, 2007, Lemma 21). We update the case for extended variables.

- The case ($s = \phi_1\hat{}\pi_1\cdot X$). Suppose $b\#X \in \nabla$ for all $b \in \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ and for each $a \in \mathtt{ds}(\nabla, \pi, \pi')$ so that $\nabla \vdash a\#\phi_1\hat{}\pi_1\cdot X$ is derivable using ($\#_{\mathbf{alt}}\mathbf{X}$).

  Now, unpacking the definition of $\mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ we observe that, for every $n \in \mathscr{A}$, if $\nabla \vdash a\#\phi_1(\pi_1(n))$ for each $a \in \mathtt{ds}(\pi, \pi')$ then, by inductive hypothesis, $\nabla \vdash$

$\pi\cdot(\phi_1(\pi_1(n))) \approx_\alpha \pi'\cdot(\phi_1(\pi_1(n)))$. Therefore, it is the case that $\mathtt{ds}(\nabla,(\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1),(\pi'\cdot\phi_1)\hat{}(\pi'\circ\pi_1)) = \bigcup_{a\in\mathtt{ds}(\nabla,\pi,\pi')} \mathtt{fresh}(\nabla,a,\phi_1\hat{}\pi_1)$, and we are able to derive $\nabla\vdash(\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1)\cdot X \approx_\alpha (\pi'\cdot\phi_1)\hat{}(\pi'\circ\pi_1)\cdot X$ using $(\approx_{\alpha\mathbf{alt}}\mathbf{X})$. The result follows by application of Definition 2.1.8.

■

The next lemma shows that α-equivalence respects the freshness relation.

Lemma 2.5.7. *If $\nabla\vdash n\#s$ and $\nabla\vdash s\approx_\alpha t$, then $\nabla\vdash n\#t$.*

Proof. By induction on the structure of s. The proof for non-extended terms can be found in (Fernández and Gabbay, 2007, Lemma 23). Below, we update the case for extended variables.

- The case $(s=\phi\hat{}\pi\cdot X)$. By the syntax-directed nature of the rules, the derivation must conclude in $(\approx_{\alpha\mathbf{alt}}\mathbf{X})$ where $t=\phi'\hat{}\pi'\cdot X$ and $a\#X\in\nabla$ for all $a\in\mathtt{ds}(\nabla,\phi\hat{}\pi,\phi'\hat{}\pi')$.

  Now suppose $b\#X\in\nabla$ for each $b\in\mathtt{fresh}(\nabla,n,\phi\hat{}\pi)$ so that $\nabla\vdash n\#\phi\hat{}\pi\cdot X$ is derivable using $(\#_{\mathbf{alt}}\mathbf{X})$. Then, we must find out which atoms are occurring in $\mathtt{fresh}(\nabla,n,\phi'\hat{}\pi')$ so that $\nabla\vdash n\#\phi'\hat{}\pi'\cdot X$ is also derivable.

  Unpacking the definition of $\mathtt{fresh}(\nabla,n,\phi\hat{}\pi)$ we observe the following: for any atom c such that $c\notin\mathtt{fresh}(\nabla,n,\phi\hat{}\pi)$, it is the case that $\nabla\vdash n\#\phi(\pi(c))$. Then, if $c\notin\mathtt{ds}(\nabla,\phi\hat{}\pi,\phi'\hat{}\pi')$, we know that $\nabla\vdash\phi(\pi(c))\approx_\alpha\phi(\pi(c))$ using $(\approx_{\alpha\mathbf{alt}}\mathbf{X})$ and, by inductive hypothesis, we have $\nabla\vdash n\#\phi'(\pi'(c))$. Therefore $c\notin\mathtt{fresh}(\nabla,n,\phi'\hat{}\pi')$. Then, it can only be that $\mathtt{fresh}(\nabla,n,\phi'\hat{}\pi')\subseteq(\mathtt{fresh}(\nabla,n,\phi\hat{}\pi)\cup\mathtt{ds}(\nabla,\phi\hat{}\pi,\phi'\hat{}\pi'))$, and the result follows by application of $(\#_{\mathbf{alt}}\mathbf{X})$.

■

We are now ready to demonstrate the main theorem of this section, that $\approx_\alpha$ is also an equivalence relation for extended nominal terms.

Theorem 2.5.8 (α-equivalence relation)**.** *$\approx_\alpha$ is an equivalence relation on nominal terms since*

1. *$\nabla\vdash s\approx_\alpha s$ (reflexive),*
2. *if $\nabla\vdash s\approx_\alpha t$ then $\nabla\vdash t\approx_\alpha s$ (symmetric) and*
3. *if $\nabla\vdash s\approx_\alpha t$ and $\nabla\vdash t\approx_\alpha u$, then $\nabla\vdash s\approx_\alpha u$ (transitive).*

Proof. We handle the three cases separately.

The reflexive case.

It is proved by an easy induction on s. We omit the proof.

The symmetric case.

By induction on the derivation of $\nabla \vdash s \approx_\alpha t$. The interesting case is that of the variable. Other cases can be found in Appendix A.

- The case ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$). Suppose $a\#X \in \nabla$ for all $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ so that $\nabla \vdash \phi\hat{}\pi{\cdot}X \approx_\alpha \phi'\hat{}\pi'{\cdot}X$ is derivable using ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$). By inductive hypothesis on the unpacked definition of $\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ we obtain $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi'(\pi'(a)) \approx_\alpha \phi(\pi(a))\}$ such that $\mathtt{ds}(\nabla, \phi'\hat{}\pi', \phi\hat{}\pi) = \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. The result follows by application of rule ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$).

The transitive case.

By induction on the derivation of $\nabla \vdash s \approx_\alpha t$, $t \approx_\alpha u$. The interesting case is that of the variable. Other cases can be found in Appendix A.

- The case ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$). Suppose $\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_t\hat{}\pi_t)\#X \subseteq \nabla, \mathtt{ds}(\nabla, \phi_t\hat{}\pi_t, \phi_u\hat{}\pi_u)\#X \in \nabla$. Then, both $\nabla \vdash \phi_s\hat{}\pi_s{\cdot}X \approx_\alpha \phi_t\hat{}\pi_t{\cdot}X$ and $\nabla \vdash \phi_t\hat{}\pi_t{\cdot}X \approx_\alpha \phi_u\hat{}\pi_u{\cdot}X$ are derivable by rule ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$). We must prove that $\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_u\hat{}\pi_u) \subseteq$ $(\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_t\hat{}\pi_t) \cup \mathtt{ds}(\nabla, \phi_t\hat{}\pi_t, \phi_u\hat{}\pi_u))$. This can be done by application of the inductive step on the unpacked definition of both $\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_t\hat{}\pi_t)$ and $\mathtt{ds}(\nabla, \phi_t\hat{}\pi_t, \phi_u\hat{}\pi_u)$, that is, $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi_s(\pi_s(a)) \approx_\alpha \phi_t(\pi_t(a))\}$ and $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi_t(\pi_t(a)) \approx_\alpha \phi_u(\pi_u(a))\}$, such that $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi_s(\pi_s(a)) \approx_\alpha \phi_u(\pi_u(a))\}$ is obtained. Then, the result follow by rule ($\approx_{\alpha\mathbf{alt}}\mathbf{X}$).

■

Having proved that $\approx_\alpha$ is indeed an equivalence relation, we turn our attention to the properties involving application of atom actions. We begin by extending the commutative property. Below, Property 2.5.9 states that permutations and a-substitutions commute. Similarly, in Property 2.5.10 we show that permutations and v-substitutions also commute. Lastly, in Property 2.5.11, the commutative property is stated with respect to a-substitutions and v-substitutions.

Property 2.5.9 (π, ϕ commute)**.**

- $\nabla \vdash \pi{\cdot}(s\phi) \approx_\alpha (\pi{\cdot}s)(\pi{\cdot}\phi)$.
- $\nabla \vdash (\pi{\cdot}s)\phi \approx_\alpha \pi{\cdot}(s(\pi^{-1}{\cdot}\phi))$.

Proof. By induction on the syntax of s on both cases. We prove both claims together.

Interesting cases are those for the variable and both abstraction cases, since abstraction cases are similar, we show the case for $(s = [a]s')$ only. Other cases can be found in Appendix A.

- The case $(s = \phi\hat{}\pi'\cdot X)$.

  For the first claim.

  Suppose $\nabla \vdash \pi\cdot((\phi'\hat{}\pi'\cdot X)\phi)$. By Definition 2.3.2, $\nabla \vdash \pi\cdot((\phi'\hat{}\pi'\cdot X)\phi) \approx_\alpha \pi\cdot((\phi' \bullet \phi)\hat{}\pi'\cdot X)$. By Definition 2.1.8, $\pi\cdot((\phi' \bullet \phi)\hat{}\pi'\cdot X) = ((\pi\cdot\phi') \bullet (\pi\cdot\phi))\hat{}(\pi \circ \pi')\cdot X$. Applying again Definition 2.3.2 we obtain $\nabla \vdash ((\pi\cdot\phi') \bullet (\pi\cdot\phi))\hat{}(\pi \circ \pi')\cdot X \approx_\alpha ((\pi\cdot\phi')\hat{}(\pi \circ \pi')\cdot X)(\pi\cdot\phi)$. Then, by Definition 2.1.8 we have, $((\pi\cdot\phi')\hat{}(\pi \circ \pi')\cdot X)(\pi\cdot\phi) = (\pi\cdot(\phi'\hat{}\pi'\cdot X))(\pi\cdot\phi)$. The result follows by the transitive property from Theorem 2.5.8.

  For the second claim.

  Suppose $\nabla \vdash (\pi\cdot(\phi'\hat{}\pi'\cdot X))\phi$. By Definition 2.1.8 we have, $(\pi\cdot(\phi'\hat{}\pi'\cdot X))\phi = ((\pi\cdot\phi')\hat{}(\pi \circ \pi')\cdot X)\phi$. By Definition 2.3.2 we have, $\nabla \vdash ((\pi\cdot\phi')\hat{}(\pi \circ \pi')\cdot X)\phi \approx_\alpha ((\pi\cdot\phi') \bullet \phi)\hat{}(\pi \circ \pi')\cdot X$. By Definition 2.1.8, $((\pi\cdot\phi') \bullet \phi)\hat{}(\pi \circ \pi')\cdot X = \pi\cdot((\phi' \bullet (\pi^{-1}\cdot\phi))\hat{}\pi'\cdot X)$. Applying again Definition 2.3.2 we obtain, $\nabla \vdash \pi\cdot((\phi' \bullet (\pi^{-1}\cdot\phi))\hat{}\pi'\cdot X) \approx_\alpha \pi\cdot((\phi'\hat{}\pi'\cdot X)(\pi^{-1}\cdot\phi))$. The result follows by the transitive property from Theorem 2.5.8.

- The case $(s = [a]s')$.

  For the first claim.

  Suppose $\nabla \vdash \pi\cdot(((a\ c)\cdot s')\phi^{-c}) \approx_\alpha (\pi\cdot((a\ c)\cdot s'))(\pi\cdot\phi^{-c})$ where $\nabla \vdash c\#s', \mathscr{I}mg(\phi)$. Using $(\approx_{\alpha_{\mathbf{alt}}[\mathbf{a}]})$ we obtain $\nabla \vdash [\pi(c)]\pi\cdot(((a\ c)\cdot s')\phi^{-c}) \approx_\alpha [\pi(c)](\pi\cdot((a\ c)\cdot s'))(\pi\cdot\phi^{-c})$. By Definition 2.1.8 on the LHS we get, $[\pi(c)]\pi\cdot(((a\ c)\cdot s')\phi^{-c}) = \pi\cdot([c]((a\ c)\cdot s')\phi^{-c})$ and on the RHS we have, $[\pi(c)](\pi\cdot((a\ c)\cdot s'))(\pi\cdot\phi^{-c}) = [\pi(c)](\pi(a\ c)\cdot s')(\pi\cdot\phi^{-c})$. Now, since $\mathtt{ds}(\nabla, \pi(a\ c), (\pi(a)\ \pi(c))\pi) = \varnothing$ then, on the RHS we have $\nabla \vdash [\pi(c)](\pi(a\ c)\cdot s')(\pi\cdot\phi^{-c}) \approx_\alpha [\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')(\pi\cdot\phi^{-c})$. By Definition 2.3.2 on the LHS, $\nabla \vdash \pi\cdot([c]((a\ c)\cdot s')\phi^{-c}) \approx_\alpha \pi\cdot(([a]s')\phi)$. By the transitive property from Theorem 2.5.8, $\nabla \vdash \pi\cdot(([a]s')\phi) \approx_\alpha [\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')(\pi\cdot\phi^{-c})$. By Definition 2.3.2 on the RHS, $\nabla \vdash [\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')(\pi\cdot\phi^{-c}) \approx_\alpha ([\pi(a)]\pi\cdot s')(\pi\cdot\phi)$. By Definition 2.1.8 on the RHS, $([\pi(a)]\pi\cdot s')(\pi\cdot\phi) = (\pi\cdot([a]s'))(\pi\cdot\phi)$. The result follows by the transitive property from Theorem 2.5.8.

  For the second claim.

  Suppose $\nabla \vdash (\pi\cdot((a\ c)\cdot s'))\phi^{-c} \approx_\alpha \pi\cdot(((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c})$ where $\nabla \vdash c\#s', \mathscr{I}mg(\phi)$. Using $(\approx_{\alpha_{\mathbf{alt}}[\mathbf{a}]})$ we obtain, $\nabla \vdash [\pi(c)](\pi\cdot((a\ c)\cdot s'))\phi^{-c} \approx_\alpha [\pi(c)]\pi\cdot(((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c})$. On the LHS we have, by Definition 2.1.8, $[\pi(c)](\pi\cdot((a\ c)\cdot s'))\phi^{-c} = [\pi(c)](\pi(a\ c)\cdot s')\phi^{-c}$. By the property of permutations we obtain, $[\pi(c)](\pi(a\ c)\cdot s')\phi^{-c} = [\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')\phi^{-c}$ and by Definition 2.1.8, $[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')\phi^{-c} = [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot s'))\phi^{-c}$. By Definition 2.3.2 on

the LHS we get $\nabla \vdash [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot s'))\phi^{-c} \approx_\alpha ([\pi(a)](\pi\cdot s'))\phi$ and by Definition 2.1.8, $([\pi(a)](\pi\cdot s'))\phi = \pi\cdot([a]s')\phi$. Now, on the RHS we have, by Definition 2.1.8, $[\pi(c)]\pi\cdot(((a\ c)\cdot s')\pi^{-1}\cdot\phi^{-c} = \pi\cdot([c]((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c}))$. By Definition 2.3.2, $\nabla \vdash \pi\cdot([c]((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c}))$. The result follows by the transitive property from Theorem 2.5.8.

■

Property 2.5.10 (π, σ Commute)**.**

- $\nabla \vdash \pi\cdot(s\sigma) \approx_\alpha (\pi\cdot s)\sigma$.

Proof. By induction on the syntax of s. The proof can be found in (Fernández and Gabbay, 2007, Lemma 5). We update the case for variables.

- The case ($s = \phi\hat{}\pi'\cdot X$). Suppose $\nabla \vdash \pi\cdot((\phi\hat{}\pi'\cdot X)\sigma)$ where $\phi = [a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]$. By application of Definition 2.3.9, $\nabla \vdash \pi\cdot((\phi\hat{}\pi'\cdot X)\sigma) \approx_\alpha \pi\cdot((\pi'\cdot\sigma(X))\phi\sigma)$ where, by Definition 2.3.8, $\phi\sigma = [a_1 \mapsto s_1\sigma; \ldots; a_n \mapsto s_n\sigma]$. By application of Property 2.5.9 on the RHS we have, $\nabla \vdash \pi\cdot((\pi'\cdot\sigma(X))\phi\sigma) \approx_\alpha (\pi\cdot(\pi'\cdot\sigma(X)))(\pi\cdot(\phi\sigma))$ where, by Definition 2.1.7, $\pi\cdot(\phi\sigma) = [\pi(a_1) \mapsto \pi\cdot(s_1\sigma); \ldots; \pi(a_n) \mapsto \pi\cdot(s_n\sigma)]$. Applying the inductive step we observe that, $\nabla \vdash \pi\cdot(s_i\sigma) \approx_\alpha (\pi\cdot s_i)\sigma$ for each $\pi\cdot(s_i\sigma) \in \mathscr{I}mg(\pi\cdot(\phi\sigma))$, $1 \leq i \leq n$ such that $\nabla \vdash (\pi\cdot(\pi'\cdot\sigma(X)))(\pi\cdot(\phi\sigma)) \approx_\alpha (\pi\cdot(\pi'\cdot\sigma(X)))((\pi\cdot\phi)\sigma)$. By application of Definition 2.1.8 we obtain, $(\pi\cdot(\pi'\cdot\sigma(X)))((\pi\cdot\phi)\sigma) = ((\pi\circ\pi')\cdot\sigma(X))((\pi\cdot\phi)\sigma)$ such that applying Definition 2.3.9 we have, $\nabla \vdash ((\pi\circ\pi')\cdot\sigma(X))((\pi\cdot\phi)\sigma) \approx_\alpha ((\pi\cdot\phi)\hat{}(\pi\circ\pi')\cdot X)\sigma$. The result follows by Definition 2.1.8 and the transitive property from Theorem 2.5.8.

■

Property 2.5.11 (ϕ, σ Commute)**.**

- $\nabla \vdash (s\sigma)\phi\sigma \approx_\alpha (s\phi)\sigma$.

Proof. By induction on the structure of s. We show the case for variables. Other cases can be found in Appendix A.

- The case ($s = \phi'\hat{}\pi\cdot X$). Suppose $\nabla \vdash ((\phi'\hat{}\pi\cdot X)\sigma)\phi\sigma$. By Definition 2.3.9 we have, $\nabla \vdash ((\phi'\hat{}\pi\cdot X)\sigma)\phi\sigma \approx_\alpha ((\pi\cdot\sigma(X))(\phi'\sigma))(\phi\sigma)$. By application of Definition 2.1.5, $((\pi\cdot\sigma(X))(\phi'\sigma))(\phi\sigma) = (\pi\cdot\sigma(X)(\phi'\sigma \bullet \phi\sigma)$. By application of Definition 2.3.2, $\nabla \vdash (\pi\cdot\sigma(X)(\phi'\sigma \bullet \phi\sigma) \approx_\alpha ((\phi'\sigma \bullet \phi\sigma)\hat{}\pi\cdot\sigma(X)$. Using the transitive property from Theorem 2.5.8, $\nabla \vdash ((\phi'\hat{}\pi\cdot X)\sigma)\phi\sigma \approx_\alpha ((\phi'\sigma \bullet \phi\sigma)\hat{}\pi\cdot\sigma(X)$. By Definition 2.3.9, $\nabla \vdash ((\phi'\sigma \bullet \phi\sigma)\hat{}\pi\cdot\sigma(X) \approx_\alpha ((\phi' \bullet \phi)\hat{}\pi\cdot X)\sigma$. The result follows by Definition 2.3.2 and the transitive property from Theorem 2.5.8.

■

The lemma below shows that freshness and α-equivalence judgements are preserved under the action of v-substitutions.

Lemma 2.5.12. *For any pair of freshness contexts Δ, ∇ and v-substitution σ such that $\Delta \vdash \nabla\sigma$,*

1. *If $\nabla \vdash a\#t$ then $\Delta \vdash a\#t\sigma$, and*
2. *If $\nabla \vdash s \approx_\alpha t$ then $\Delta \vdash s\sigma \approx_\alpha t\sigma$.*

Proof. We handle both cases separately. Interesting cases are those of variables. The rest can be found in Appendix A.

For the first claim.

By induction on the derivation of $\nabla \vdash a\#t$.

- The case ($\#_{\mathbf{alt}}\mathbf{X}$). Suppose that $b\#X \in \nabla$ for all $b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ so that $\nabla \vdash a\#\phi\hat{}\pi\cdot X$ is derivable by ($\#_{\mathbf{alt}}\mathbf{X}$). Then, by case (4) of Lemma 2.5.4 we obtain $\nabla \vdash b\#X$ and by the assumptions, $\Delta \vdash b\#X\sigma$. Using the inductive hypothesis on the unpacked definition of $\mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ we obtain $\{b \mid b \in \mathscr{A} \wedge \Delta \nvdash a\#(\phi(\pi(b))\sigma)\}$ such that $\mathtt{fresh}(\Delta, a, (\phi\sigma)\hat{}\pi)$ where $\mathtt{fresh}(\Delta, a, (\phi\sigma)\hat{}\pi) \subseteq \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$. Using ($\#_{\mathbf{alt}}\mathbf{X}$) we derive $\nabla \vdash a\#(\pi\cdot\sigma(X))(\phi\sigma)$ and the result follows by Definition 2.3.9.

For the second claim.

By induction on the derivation of $\nabla \vdash s \approx_\alpha t$.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$). Suppose $a\#X \in \nabla$ for all $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. By case (4) of Lemma 2.5.4 we have $\nabla \vdash a\#X$ and applying the first claim of this lemma we show that $\Delta \vdash a\#\sigma(X)$. Now we apply the inductive hypothesis on the definition of $\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$: $\{a \mid a \in \mathscr{A} \wedge \Delta \nvdash (\phi(\pi(a)))\sigma \approx_\alpha (\phi'(\pi'(a)))\sigma\}$ and we are able to construct $\mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$ where $\mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi') \subseteq \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. The result follows by application of ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$) and Definition 2.3.9.

■

The following lemma states that application of α-equivalent v-substitutions on a term preserve α-equivalence between both instances.

Lemma 2.5.13. *If $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all $X \in V(s)$, then $\nabla \vdash s\sigma \approx_\alpha s\sigma'$.*

Proof. By induction on the syntax of s. See Appendix A. ■

Next, we provide an extension of Lemma 2.5.6 to accommodate a-substitutions. It states that distinct atom operations acting on the same term s produce α-equivalent terms when the difference set is fresh for s.

Lemma 2.5.14. *If $\nabla \vdash n\#s$ for each $n \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$, then $\nabla \vdash (\pi\cdot s)\phi \approx_\alpha (\pi'\cdot s)\phi'$.*

Proof. By induction on the structure of s. We show the case for variables and abstraction terms; other cases can be found in Appendix A.

- The case ($s = \phi_1\hat{}\pi_1\cdot X$). Let $b\#X \in \nabla$ for each $b \in \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$, where $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ such that $\nabla \vdash a\#\phi_1\hat{}\pi_1\cdot X$ is derivable using ($\#_{\mathbf{alt}}\mathbf{X}$). Unpacking the definition of $\mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ we observe that $\nabla \vdash a\#\phi_1(\pi_1(n))$ for all $n \in \mathscr{A} \wedge n \notin \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ and by application of the inductive hypothesis we obtain, $\nabla \vdash (\pi\cdot(\phi_1(\pi_1(n))))\phi \approx_\alpha (\pi'\cdot(\phi_1(\pi_1(n))))\phi'$. Hence, $\mathtt{ds}(\nabla, ((\pi\cdot\phi_1)\bullet\phi)\hat{}(\pi\circ\pi_1), ((\pi'\cdot\phi_1)\bullet\phi')\hat{}(\pi'\circ\pi_1)) = \bigcup_{a\in\mathtt{ds}(\nabla,\phi\hat{}\pi,\phi'\hat{}\pi')} \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ and using ($\approx_{\alpha\,\mathbf{alt}}\mathbf{X}$) we derive $\nabla \vdash ((\pi\cdot\phi_1)\bullet\phi)\hat{}(\pi\circ\pi_1)\cdot X \approx_\alpha ((\pi'\cdot\phi_1)\bullet\phi')\hat{}(\pi'\circ\pi_1)\cdot X$. By Definition 2.3.2 we have, on the LHS, $\nabla \vdash ((\pi\cdot\phi_1)\bullet\phi)\hat{}(\pi\circ\pi_1)\cdot X \approx_\alpha ((\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1)\cdot X)\phi$ and, on the RHS, $\nabla \vdash ((\pi'\cdot\phi_1)\bullet\phi')\hat{}(\pi'\circ\pi_1)\cdot X \approx_\alpha ((\pi'\cdot\phi_1)\hat{}(\pi'\circ\pi_1)\cdot X)\phi'$. By the transitive property from Theorem 2.5.8 we obtain $\nabla \vdash ((\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1)\cdot X)\phi \approx_\alpha ((\pi'\cdot\phi_1)\hat{}(\pi'\circ\pi_1)\cdot X)\phi'$. The result follows by Definition 2.1.8.

- The case ($s = [a]t$). We observe that either $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ or not. However, both cases are resolved similarly because of the action of a-substitutions on abstractions which requires choosing distinct representatives on both sides of the equation prior application of a-substitutions. Then, $\mathtt{ds}(\nabla, ((c\ c')\cdot\phi)\hat{}(c\ c')(\pi(a)\ c)\pi, \phi'\hat{}(\pi'(a)\ c')\pi')$ is applied to either case and the proof is solved as follows.

  Suppose $\nabla \vdash c, c'\#t, \mathscr{I}mg(\phi), \mathscr{I}mg(\phi')$. Suppose also that $\nabla \vdash n\#t$ for each $n \in \mathtt{ds}(\nabla, ((c\ c')\cdot\phi^{-c})\hat{}(c\ c')(\pi(a)\ c)\pi, \phi'^{-c'}\hat{}(\pi'(a)\ c')\pi')$. By inductive hypothesis, $\nabla \vdash ((c\ c')(\pi(a)\ c)\pi\cdot t)((c\ c')\cdot\phi^{-c}) \approx_\alpha ((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'}$. Using Definition 2.1.8 we have $((c\ c')(\pi(a)\ c)\pi\cdot t)((c\ c')\cdot\phi^{-c}) = ((c\ c')\cdot((\pi(a)\ c)\pi\cdot t))((c\ c')\cdot\phi^{-c})$. By the first claim of the commutative property from Property 2.5.9, $\nabla \vdash ((c\ c')\cdot((\pi(a)\ c)\pi\cdot t))((c\ c')\cdot\phi^{-c}) \approx_\alpha (c\ c')\cdot(((\pi(a)\ c)\pi\cdot t)\phi^{-c})$. Then, by the transitive property from Theorem 2.5.8 we obtain $\nabla \vdash (c\ c')\cdot(((\pi(a)\ c)\pi\cdot t)\phi^{-c}) \approx_\alpha ((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'}$. Using ($\approx_{\alpha\,\mathbf{alt}}[\mathbf{b}]$) we derive $\nabla \vdash [c]((\pi(a)\ c)\pi\cdot t)\phi^{-c}) \approx_\alpha [c']((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'}$. Then, by Definition 2.1.8, $[c]((\pi(a)\ c)\pi\cdot t)\phi) = [c]((\pi(a)\ c)\cdot(\pi\cdot t))\phi)$. Applying Definition 2.3.2 we have, on the LHS, $\nabla \vdash [c]((\pi(a)\ c)\cdot(\pi\cdot t))\phi^{-c}) \approx_\alpha ([\pi(a)]\pi\cdot t)\phi)$, and on the RHS, $\nabla \vdash [c']((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'} \approx_\alpha ([\pi'(a)]\pi'\cdot t)\phi'$. By Definition 2.1.8, $([\pi(a)]\pi\cdot t)\phi) = (\pi\cdot[a]t)\phi)$ on the LHS and $([\pi'(a)]\pi'\cdot t)\phi' = (\pi'\cdot([a]t))\phi'$ on the RHS. The result follows by the transitive property from Theorem 2.5.8.

■

The lemma that follows is the converse of the result provided in the lemma above. It will be useful later on, when providing properties of the unification problem.

Lemma 2.5.15. *If* $\nabla \vdash (\pi{\cdot}s)\phi \approx_\alpha (\pi'{\cdot}s)\phi'$ *then* $\nabla \vdash n\#s$ *for each* $n \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$.

Proof. By induction on the syntax of s. We show the case for abstraction terms; other cases are found in Appendix A.

- The case $(s = [a]s)$. Suppose $\nabla \vdash ((c\ c')(c\ \pi(a))\pi{\cdot}s)((c\ c'){\cdot}\phi^{-c}) \approx_\alpha$ $((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'}$ where $\nabla \vdash c, c'\#s, \mathscr{I}mg(\phi), \mathscr{I}mg(\phi')$. Then, by the inductive hypothesis, $\nabla \vdash n\#s$ for each $n \in \mathtt{ds}(\nabla, ((c\ c'){\cdot}\phi^{-c})\hat{}(c\ c')(c\ \pi(a))\pi, \phi'^{-c'}\hat{}(c'\ \pi'(a))\pi')$ where
  $\mathtt{ds}(\nabla, ((c\ c'){\cdot}\phi^{-c})\hat{}(c\ c')(c\ \pi(a))\pi, \phi'^{-c'}\hat{}(c'\ \pi'(a))\pi') = \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi') \setminus \{c, c'\}$.
  Using $(\#_{\mathbf{alt}}[\mathbf{a}])$ or $(\#_{\mathbf{alt}}[\mathbf{b}])$ we derive $\nabla \vdash n\#[a]s$.

  By the second claim from Property 2.5.9, we have

  $\nabla \vdash (((c\ c')(c\ \pi(a))\pi{\cdot}s)((c\ c'){\cdot}\phi^{-c}) \approx_\alpha (c\ c'){\cdot}((c\ \pi(a))\pi{\cdot}s)\phi^{-c'})$. Using the transitive property from Theorem 2.5.8 we obtain $\nabla \vdash (c\ c'){\cdot}(((c\ \pi(a))\pi{\cdot}s)\phi^{-c}) \approx_\alpha ((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'}$. By using $(\approx\alpha_{\mathbf{alt}}[\mathbf{b}])$ we derive $\nabla \vdash [c](((c\ \pi(a))\pi{\cdot}s)\phi^{-c}) \approx_\alpha [c'](((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'}$. By Definition 2.3.2 we have, on the LHS,
  $\nabla \vdash [c](((c\ \pi(a))\pi{\cdot}s)\phi^{-c}) \approx_\alpha ([\pi(a)]\pi{\cdot}s)\phi$ and on the RHS,
  $\nabla \vdash [c'](((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'} \approx_\alpha ([\pi'(a)]\pi'{\cdot}s)\phi'$. Applying the transitive property from Theorem 2.5.8, $\nabla \vdash ([\pi(a)]\pi{\cdot}s)\phi) \approx_\alpha ([\pi'(a)]\pi'{\cdot}s)\phi'$. The result follows by Definition 2.1.8.

■

The lemma below is similar to Lemma 2.5.14, but in this case applied to the freshness relation.

Lemma 2.5.16. *If* $\nabla \vdash n\#s$ *for each* $n \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$, *then* $\nabla \vdash a\#(\pi{\cdot}s)\phi$.

Proof. By induction on the structure of s. We show the variable and the abstraction term; other cases are found in Appendix A.

- The case $(s = \phi'\hat{}\pi'{\cdot}X)$. Let $b\#X \in \nabla$ for each $b \in \mathtt{fresh}(\nabla, n, \phi'\hat{}\pi')$ and each $n \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ such that $\nabla \vdash n\#\phi'\hat{}\pi'{\cdot}X$ is derivable using $(\#_{\mathbf{alt}}\mathbf{X})$. Unpacking the definition of $\mathtt{fresh}(\nabla, n, \phi'\hat{}\pi')$ we observe that $\nabla \vdash n\#\phi'(\pi'(c))$ for all $c \in (\mathscr{A} \setminus \mathtt{fresh}(\nabla, n, \phi'\hat{}\pi'))$, and by application of the inductive hypothesis we obtain $\nabla \vdash a\#(\pi{\cdot}(\phi'(\pi'(c))))\phi$. Hence, $\mathtt{fresh}(\nabla, a, ((\pi{\cdot}\phi') \bullet \phi)\hat{}(\pi \circ \pi')) = \mathtt{fresh}(\nabla, n, \phi'\hat{}\pi') \cup \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$, and using $(\#_{\mathbf{alt}}\mathbf{X})$ we derive $\nabla \vdash a\#((\pi{\cdot}\phi') \bullet \phi)\hat{}(\pi \circ \pi'){\cdot}X$. By Definition 2.3.2 we have $\nabla \vdash a\#((\pi{\cdot}\phi')\hat{}(\pi \circ \pi'){\cdot}X)\phi$. The result follows by Definition 2.1.8.

- The case $(s = [a]t)$. Either $a \in \texttt{fresh}(\nabla, a, \phi\hat{}\pi)$ or not. Observe, however, that due to the action of a-substitutions on abstractions, both cases are resolved similarly. Hence, we construct a derivation as follows: Suppose $\nabla \vdash n\#t$ for each $n \in \texttt{fresh}(\nabla, a, \phi^{-\pi(c)}\hat{}(\pi(a)\ \pi(c))\pi)$ where $\nabla \vdash \pi(c)\#\pi\cdot t, \mathscr{I}mg(\phi)$. Note that, by some basic group theory it is verifiable that $\texttt{fresh}(\nabla, a, \phi^{-\pi(c)}\hat{}(\pi(a)\ \pi(c))\pi) = \texttt{fresh}(\nabla, a, \phi^{-\phi(c)}\hat{}\pi) \setminus \{a\}$. Then, by inductive hypothesis it follows that, $\nabla \vdash a\#((\pi(a)\ \pi(c))\pi\cdot t)\phi^{-\pi(c)}$. Using (#[**b**]) we obtain, $\nabla \vdash a\#[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot t)\phi^{-\pi(c)}$. By Definition 2.1.8, $[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot t)\phi = [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot t))\phi^{-\pi(c)}$. By Definition 2.3.2, $[\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot t))\phi^{-\pi(c)} \approx_\alpha ([\pi(a)]\pi\cdot t)\phi$. The result follows by Definition 2.1.8.

■

The Lemma below is the converse of Lemma 2.5.16 and will be of use later, when stating properties in the unification section.

Lemma 2.5.17. *If $\nabla \vdash a\#(\pi\cdot s)\phi$ then $\nabla \vdash n\#s$ for each $n \in \texttt{fresh}(\nabla, a, \phi\hat{}\pi)$.*

Proof. By induction on the structure of s. See Appendix A. ■

In the following lemma we show that α-equivalence is preserved under a-substitution application. This is the case because α-equivalent terms have the same support.

Lemma 2.5.18 (ϕ and $\approx_\alpha$)**.** *Let s,t be any pair of terms and ∇ a freshness context such that $\nabla \vdash s \approx_\alpha t$. Let ϕ be an a-substitution. Then, $\nabla \vdash s\phi \approx_\alpha t\phi$.*

Proof. By induction on the derivation of $\nabla \vdash s \approx_\alpha t$. See Appendix A. ■

Theorem 2.5.19 (Atom actions and $\approx_\alpha$)**.** *If $\nabla \vdash s \approx_\alpha t$ then $\nabla \vdash (\pi\cdot s)\phi \approx_\alpha (\pi\cdot t)\phi$.*

Proof. By induction on the derivation of $\nabla \vdash s \approx_\alpha t$. The result follows by Lemmas 2.5.5 & 2.5.18. ■

Corollary 2.5.20. *α-equality is a congruence relation therefore, if $\nabla \vdash t \approx_\alpha t'$ then $\nabla \vdash s[t]_p \approx_\alpha s[t']_p$.*

Proof. By induction on the syntax of s and the application of the rules in Definition 2.2.5. The result follows by the fact that $\approx_\alpha$ is an equivalence relation (see Theorem 2.5.8) preserved under the application of atom actions (see Theorem 2.5.19). ■

2.6 Conclusion

This chapter introduced the syntax of nominal terms extended with implicit substitution and presented a set of deterministic, syntax-directed natural deduction rules to describe the relation on extended terms determined by predicates # (Definition 2.2.3) and $\approx_\alpha$ (Definition 2.2.5). α-equality between extended terms is handled elegantly through the use of a swapping operation and a freshness relation along with an explicit version of α-conversion for the semantics of atom substitutions when applied to abstraction terms (Definition 2.3.2). It was also determined that the behaviour of atom substitution corresponds with that of explicit substitution for standard nominal terms as given in Definition 1.5.4 (see Theorem 2.3.6). Finally, it was shown that the properties of standard nominal terms extend naturally to extended terms, in particular, we showed that $\approx_\alpha$ is indeed an equivalence relation on extended nominal terms since it is reflexive, symmetric and transitive (Theorem 2.5.8) and that such relation is preserved under the action of atom actions (Theorem 2.5.19).

Now, we are ready to transform the logical presentation of freshness and α-equality relations into an algorithm to check derivability of constraints.

Chapter 3

An Algorithm to Check Constraints

This chapter presents a *constraint checking algorithm* for extended nominal terms following the set of freshness and α-equality core inference rules given in Chapter 2.

The transformation of the core inference rules from a logical representation to an algorithmic one is non-trivial due to the complexities that arise when dealing with disjuncts during the derivation of the variable occurrence case because of the suspended atom substitutions. Unlike its standard counterpart (Urban et al., 2004), the normal form of an *extended constraint problem* (that is, a conjunction of freshness and α-equality constraints) is a disjunction of conjunctions (DNF) of constraints which may contain not only one or more distinct disjuncts providing evidence for the entailment relation to hold but also disjuncts which have *inconsistent constraints* or *clashing equalities* (see Definition 3.2.2), that is, redundant results. This is due to the naive generation of disjuncts during the application of the constraint checking algorithm to variable occurrences leading to the inclusion of such redundant results in its normal form. As a result, we also present a variation of the constraint checking algorithm where any disjunct containing redundant results is discarded during execution, thus generating a DNF where each sequence of conjunctions is a least freshness context entailing the derivation of the constraint problem given as input.

There are three main sections in this chapter. The first section defines the notion of a problem for extended constraints and presents definitions to operate on the syntax of problems. Next, the constraint checking algorithm is introduced, some examples of application are given and its correctness asserted. In the third section, we provide an auxiliary function to derive results from variable occurrences. Then, an enhanced constraint checking algorithm is defined on top of the previous one. Correctness results and examples are also produced for this case. The chapter ends with a summary of results in the conclusion section.

3.1 Extended Constraint Problems

We begin by describing the structure of constraint formulæ and provide a tool to enumerate subformulæ of a formula.

Definition 3.1.1 (Formula formation and subformulæ)**.** Let C range over constraints generated by the grammar in Definition 2.2.1. Then, constraint formula F is generated by the grammar:

$$F ::= C \mid (C \wedge C) \mid (C \vee C)$$

where operator $\wedge$ binds tighter that operator $\vee$.

The **set of subformulæ of a constraint formula** F is the smallest set $\mathbf{S}(F)$ satisfying the following conditions:

1. $F \in \mathbf{S}(F)$;

2. if $(C_1 * C_2) \in \mathbf{S}(F)$, then $C_1, C_2 \in \mathbf{S}(F)$ where $*$ is binary operator $\wedge$ or $\vee$.

Then, say a formula F' **is a subformula of** (a formula) F when $F' \in \mathbf{S}(F)$.

Example 3.1.2. The formula $F = ((((\mathtt{a\#X} \wedge \mathtt{a\#Y}) \wedge \top) \vee \bot) \vee \mathtt{f(a,b)} \approx_\alpha \mathtt{g(b,c)})$ has the following set of subformulæ,

$$\mathbf{S}(F) = \{((((\mathtt{a\#X} \wedge \mathtt{a\#Y}) \wedge \top) \vee \bot) \vee \mathtt{f(a,b)} \approx_\alpha \mathtt{g(b,c)}), (((\mathtt{a\#X} \wedge \mathtt{a\#Y}) \wedge \top) \vee \bot), \mathtt{f(a,b)} \approx_\alpha \mathtt{g(b,c)}, ((\mathtt{a\#X} \wedge \mathtt{a\#Y}) \wedge \top), \bot, (\mathtt{a\#X} \wedge \mathtt{a\#Y}), \top, \mathtt{a\#X}, \mathtt{a\#Y}\}.$$

The proliferation of parentheses may add difficulty on reading a complex formula. In the sequel, we alleviate the use of parentheses by using the binding precedence given to operators $\wedge, \vee$ to abbreviate formulæ. Then, formula F from Example 3.1.2 can be simplified to $\mathtt{a\#X} \wedge \mathtt{a\#Y} \wedge \top \vee \bot \vee \mathtt{f(a,b)} \approx_\alpha \mathtt{g(b,c)}$.

Below, we define the form of constraint problems suitable for the constraint checking algorithm to be introduced in Section 3.2.

Definition 3.1.3 (Extended constraint problem)**.** An **extended constraint problem** Pr, or just **problem**, is an arbitrary conjunction of freshness and α-equality constraints, $C_1, \ldots, C_n$ where symbol ‘, ‘ represents logical connective $\wedge$.

Write $\nabla \vdash C_1, \ldots, C_n$ when proofs of $\nabla \vdash C_i$ exist using the core derivation rules from Definition 2.2.3 and Definition2.2.5 and elements of the freshness context ∇ as evidence, for $1 \leq i \leq n$.

In the sequel, we use a comma to represent logical connective $\wedge$ whenever the interpretation is clear from the context.

It is now suitable to extend the notation for the functions returning the set of variables (resp. atoms) occurring in a term as well as the operator for term occurrence, so that it takes into account newly added definitions.

Definition 3.1.4. Functions $V(\cdot)$ and $A(\cdot)$ are extended for constraints C, freshness contexts ∇, problems Pr, and terms-in-context, $\nabla \vdash s$, as follows.

- $V(s \approx_\alpha t) = V(s) \cup V(t)$ and $A(s \approx_\alpha t) = A(s) \cup A(t)$;
- $V(a\#s) = V(s)$ and $A(a\#s) = A(s) \cup \{a\}$;
- $V(Pr) = \bigcup_i V(C_i)$ and $A(Pr) = \bigcup_i A(C_i)$ where $C_i \in \mathbf{S}(Pr)$ for $1 \leq i \leq n$;
- $V(\nabla) = \{X \mid a\#X \in \nabla\}$ and $A(\nabla) = \{a \mid a\#X \in \nabla\}$;
- $V(\nabla \vdash s) = V(\nabla) \cup V(s)$ and $A(\nabla \vdash s) = A(\nabla) \cup A(s)$.
- $V_{RHS}(Pr) = \bigcup_i V(t_i)$ where $s_i \approx_\alpha t_i \in \mathbf{S}(Pr)$ for $1 \leq i \leq n$;

The last bullet point constructs the union of the set of variables occurring on the RHS term of each α-equality constraint in a problem. It will be of assistance in Chapter 5 when defining a *matching algorithm*.

Say a term s **occurs in (the syntax of) a problem** Pr, $s \lhd Pr$ if C is a freshness or α-equality constraint in Pr, namely, $C \in \mathbf{S}(Pr)$, and $s \lhd C$ as defined in Definition 2.1.15.

We conclude with the corollary below, showing that entailment of a problem, $\nabla \vdash Pr$, is preserved under application of v-substitutions and permutations.

Corollary 3.1.5. *Let Pr be a problem as defined in Definition 3.1.3. For any pair of freshness contexts ∇, Δ and v-substitution σ such that $\Delta \vdash \nabla\sigma$,*

- *If $\nabla \vdash Pr$ then $\Delta \vdash Pr\sigma$ where $Pr\sigma$ is the pointwise application of σ in Pr;*
- *$\nabla \vdash Pr \Longleftrightarrow \nabla \vdash \pi \cdot Pr$ where $\pi \cdot Pr$ is the pointwise application of π to Pr.*

Proof. The first part follows directly from Lemma 2.5.12 on each freshness and α-equality constraint in Pr.

The second part follows directly from Lemma 2.5.5 on each freshness and α-equality constraint in Pr. ■

Intuitively, the second claim above states that the validity of a constraint problem is invariant under permuting atoms, an essential property of nominal theories and nominal terms. This property is known as (object-level) *equivariance*. An equivariant property operating at the meta-level will be discussed later, in Chapter 6, when defining a theory of rewriting with extended terms.

3.2 An Algorithmic Presentation of the Core Rules

3.2.1 Constraint checking algorithm

The layout of the reduction rules is a syntax-directed, *bottom-up* form of the core rules[1], that is, for any given freshness or α-equality constraint C, the constraint checking algorithm recursively decomposes the structure of C into strictly simpler constraints (this claim is proved later, in Lemma 3.2.5) by non-deterministic application of the reduction rules. However, unlike the standard derivability algorithm depicted in (Urban et al., 2004), the constraint checking algorithm for extended nominal terms may resolve to more than one least set of primitive constraints entailing C (see Example 2.2.4 for rule ($\#\mathbf{X}$) and Example 2.2.6 for rule ($\approx_\alpha$)), as a consequence of the disjunctive nature of the premise in core rules ($\#\mathbf{X}$) and ($\approx_\alpha \mathbf{X}$) and the lack of a freshness context when applying the reduction rules.

Reduction rules below expand on those originally given for nominal terms in (Urban et al., 2004). The set of rules described below transforms, non-deterministically, a constraint problem Pr into a finite sequence of disjuncts $Pr_1 \vee \ldots \vee Pr_n$. This is achieved by normalising the resulting formula into its *disjunctive normal form* (DNF).

Definition 3.2.1 (Derivability algorithm)**.** In the rules below, assume a, b denote distinct atoms, $\phi\hat{}\pi\cdot X$ an extended moderated variable and f a term former. Additionally, Pr is an arbitrary problem, $\top$ asserts that a relation holds and $\mathtt{DNF}(\cdot)$ is the normalisation of formula into DNF.

[1] we identify the bottom of a core rule with its conclusion and the top with its premises.

$$
\begin{array}{lrcl}
(\approx_\alpha \mathbf{a}) & a \approx_\alpha a, Pr & \Longrightarrow & Pr \\
(\approx_\alpha [\mathbf{a}]) & [a]s \approx_\alpha [a]t, Pr & \Longrightarrow & s \approx_\alpha t, Pr \\
(\approx_\alpha [\mathbf{b}]) & [a]s \approx_\alpha [b]t, Pr & \Longrightarrow & (b\ a)\cdot s \approx_\alpha t\ ,\ b\#s, Pr \\
(\approx_\alpha \mathbf{f}) & fs \approx_\alpha ft, Pr & \Longrightarrow & s \approx_\alpha t, Pr \\
(\approx_\alpha \mathbf{tupl}) & (s_1,\ldots,s_n) \approx_\alpha (t_1,\ldots,t_n), Pr & \Longrightarrow & s_1 \approx_\alpha t_1, \ldots, s_n \approx_\alpha t_n, Pr \\
(\approx_\alpha \mathbf{X}) & \phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot X, Pr & \Longrightarrow & \\
 & \multicolumn{3}{l}{\texttt{DNF}\Big(\bigwedge_{a\in(\mathscr{D}omP(\phi,\phi')\cup\mathscr{S}upportP(\pi,\pi'))} (\phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)) \vee a\#X), Pr\ \Big)} \\
(\#\mathbf{ab}) & a\#b, Pr & \Longrightarrow & Pr \\
(\#[\mathbf{a}]) & a\#[a]s, Pr & \Longrightarrow & Pr \\
(\#[\mathbf{b}]) & a\#[b]s, Pr & \Longrightarrow & a\#s, Pr \\
(\#\mathbf{f}) & a\#fs, Pr & \Longrightarrow & a\#s, Pr \\
(\#\mathbf{tupl}) & a\#(s_1,\ldots,s_n), Pr & \Longrightarrow & a\#s_1, \ldots, a\#s_n, Pr \\
(\#\mathbf{X}) & a\#\phi\hat{}\pi\cdot X, Pr & \Longrightarrow & \\
 & \multicolumn{3}{l}{\texttt{DNF}\Big(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} (a\#\phi(b) \vee \pi^{-1}(b)\#X), Pr\ \Big)\ (\text{where } \phi \neq \texttt{Id} \text{ or } \pi \neq \texttt{Id})}
\end{array}
$$

The set of rules are described in no particular order. They define a reduction relation $\Longrightarrow$ on problems Pr.

- Write $Pr \Longrightarrow Pr'$, or $Pr \overset{\Omega}{\Longrightarrow} Pr'$ if Pr' is obtained from Pr by application of one α-equality or one freshness reduction rule Ω. Then, $Pr \Longrightarrow_* Pr'$ denotes the reflexive and transitive closure of the one-step reduction $Pr \Longrightarrow Pr'$.

The side notation on rule (#**X**) avoids infinite recursive calls on variables.

Given a problem Pr, the constraint checking algorithm applies, non-deterministically, as many $\Longrightarrow$ as possible until no further transformations can be done, returning a DNF of irreducible constraints $Pr_1 \vee \cdots \vee Pr_n$, where each conjunction Pr_i contains a combination of constraints of form $s \approx_\alpha t$, $a\#X$, $a\#a$ and $\top$, for $1 \leq i \leq n$. Hence,

$$Pr \Longrightarrow_* Pr_1 \vee \cdots \vee Pr_n.$$

Definition 3.2.2. Say a freshness constraint is **reduced** when it is of the form $a\#X$ or $a\#a$ or $\top$ or also $\bot$. Then, call a reduced constraint **consistent** when it is not of form $a\#a$ or $\bot$, otherwise it is **inconsistent**.

Additionally, an α-equality constraint $s \approx_\alpha t$ is **clashing** when s,t have different term constructors at the root, different atoms, different moderated variables or function applic-

ations with different term formers. For instance, $\mathtt{a} \approx_\alpha \mathtt{b}, (\mathtt{s},\mathtt{t}) \approx_\alpha [\mathtt{a}]\mathtt{t}', \phi\hat{}\pi\cdot\mathtt{X} \approx_\alpha \phi'\hat{}\pi'\cdot\mathtt{Y}$, $\mathtt{f}(\mathtt{t}) \approx_\alpha \mathtt{g}(\mathtt{s})$ are all *clashing α-equalities*.

Remark 3.2.3. We are not working only on the strict syntactical structure of freshness and α-equality constraints when stating claims or providing examples. Logical operators $\wedge, \vee$ are associative, commutative and idempotent. Also, constraint $\top$ is the identity of $\wedge$ and constraint $\bot$ is the identity of $\vee$. In the sequel, we will be working with these properties without mention them.

An example of the reduction relation follows.

Example 3.2.4. The α-equality constraint $[\mathtt{a}][\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b}][\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X}$ reduces to the DNF $\top \vee \mathtt{b\#X} \vee \mathtt{a\#X} \vee (\mathtt{a\#X}, \mathtt{b\#X})$ by application of the reduction rules from Definition 3.2.1.

$\bullet\ [\mathtt{a}][\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b}][\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X} \overset{(\approx_\alpha[\mathbf{b}])}{\Longrightarrow} [\mathtt{a} \mapsto \mathtt{b}]\hat{}(\mathtt{a}\ \mathtt{b})\cdot\mathtt{X} \approx_\alpha [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X}$

$\overset{(\approx_\alpha\mathbf{X})}{\Longrightarrow}_*\ \mathtt{DNF}((\mathtt{b} \approx_\alpha \mathtt{b} \vee \mathtt{a\#X}), (\mathtt{b} \approx_\alpha \mathtt{b} \vee \mathtt{b\#X}), \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X})$

$= (\mathtt{b} \approx_\alpha \mathtt{b}, \mathtt{b} \approx_\alpha \mathtt{b}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X}) \vee (\mathtt{b} \approx_\alpha \mathtt{b}, \mathtt{b\#X}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X}) \vee (\mathtt{a\#X}, \mathtt{b} \approx_\alpha \mathtt{b}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X})$

$\vee(\mathtt{a\#X}, \mathtt{b\#X}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X})$ (by DNF normalisation)

$\overset{(\approx_\alpha\mathbf{a})}{\Longrightarrow}_*\ (\top, \top, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X}) \vee (\top, \mathtt{b\#X}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X}) \vee (\top, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X})$

$\vee(\mathtt{a\#X}, \mathtt{b\#X}, \mathtt{b\#}[\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X})$

$\overset{(\#\mathbf{X})}{\Longrightarrow}_*\ \mathtt{DNF}(\mathtt{b\#a} \vee \mathtt{b\#X}) \vee \mathtt{DNF}(\mathtt{b\#X}, (\mathtt{b\#a} \vee \mathtt{b\#X})) \vee \mathtt{DNF}(\mathtt{a\#X}, (\mathtt{b\#a} \vee \mathtt{b\#X}))$

$\vee\mathtt{DNF}(\mathtt{a\#X}, \mathtt{b\#X}, (\mathtt{b\#a} \vee \mathtt{b\#X}))$

$= \mathtt{b\#a} \vee \mathtt{b\#X} \vee (\mathtt{b\#X}, \mathtt{b\#a}) \vee (\mathtt{b\#X}, \mathtt{b\#X}) \vee (\mathtt{a\#X}, \mathtt{b\#a}) \vee (\mathtt{a\#X}, \mathtt{b\#X}) \vee (\mathtt{a\#X}, \mathtt{b\#X}, \mathtt{b\#a})$

$\vee(\mathtt{a\#X}, \mathtt{b\#X}, \mathtt{b\#X})$ (by DNF normalisation)

$\overset{(\#\mathbf{ab})}{\Longrightarrow}_*\ \top \vee \mathtt{b\#X} \vee (\mathtt{b\#X}, \top) \vee (\mathtt{b\#X}, \mathtt{b\#X}) \vee (\mathtt{a\#X}, \top) \vee (\mathtt{a\#X}, \mathtt{b\#X}) \vee (\mathtt{a\#X}, \mathtt{b\#X}, \top)$

$\vee(\mathtt{a\#X}, \mathtt{b\#X}, \mathtt{b\#X})$

$= \top \vee \mathtt{b\#X} \vee \mathtt{a\#X} \vee (\mathtt{a\#X}, \mathtt{b\#X})$.

3.2.2 Properties of the reduction rules

The section continues by showing relation $\Longrightarrow$ to be *convergent*, that is, terminating and confluent.

Below, we demonstrate that the set of freshness and α-equality transformation rules induces a terminating sequence of reductions on the class of extended terms. The issue here is that some reductions increase the length of the formula on the RHS, particularly the variable case because of the normalisation by DNF. However, one can define an interpretation for predicate symbols where $\approx_\alpha$ is the highest order and use a multiset order to show that at each step there is a strictly smaller problem either by the size of each constraint or by the interpretation order given to the predicates, possibly both.

Lemma 3.2.5. *There is no infinite sequence of reductions for the relation defined by the reduction rules in Definition 3.2.1.*

Proof. For an arbitrary problem Pr we show that non-deterministic application of freshness and α-equality reduction rules to any Pr terminates. To build the termination proof we take into account that some reduction rules, when applied, increase the length of the formula on the RHS of $\Longrightarrow$. Moreover, both reduction rules (#**X**), ($\approx_\alpha$**X**) are *duplicating* due to the normalisation of the formula by DNF, that is, there exists a problem Pr' on the LHS of both variable rules that is repeated along each of the disjuncts originated on the DNF formula of the RHS of $\Longrightarrow$.

To prove termination of the sequence of reductions taking into account these factor we rely on the use of multiset orderings (Dershowitz and Manna, 1979) as follows. Define $\tau : F \to \mathscr{M}(<\mathbb{N},\mathbb{N}>)$ as the termination function that is applied to any formulae over the class of extended nominal terms whose components are constraints of form $s \approx_\alpha t, a\#t$ or $\top$ with connectives $\wedge,\vee$, returning a multiset ordering of pairs $< pred, |C| >$ over the natural numbers where $< pred, |C| >=< 2, |s|+|t| >$ if C has form $s \approx_\alpha t$, $< pred, |C| >=< 1, |t| >$ if C has form $a\#t$ and $< 0,0 >$ if $pred = \top$ with $|\cdot|$ the size of an extended term following Definition 2.1.13.

Using function τ we demonstrate that the multiset interpretation is strictly decreasing, $\gg$, when applying any of the reduction rules described in Definition 3.2.1 to Pr, that is, $\tau(Pr) \gg \tau(Pr')$ where $Pr' \Longrightarrow Pr$. $\tau(Pr')$ is strictly smaller than $\tau(Pr)$ if, after one step application of some rule from the set of reduction rules, a constraint C in Pr with $< pred_C, |C| >$ is replaced by one or more constraints C'_i with $< pred_{C'_i}, |C_i| >$ such that $(pred_C > pred_{C'_i})$ or $(pred_C = pred_{C'_i})$ and $(|C| > |C'_i|)$ for each new constraint C'_i in Pr'.

Hence, for rules ($\approx_\alpha$**a**), (#**ab**) and (#[**a**]) the property holds trivially since each constraint is replaced by $\top$ on the RHS of $\Longrightarrow$. For rules ($\approx_\alpha$[**a**]), ($\approx_\alpha$**f**), (#[**b**]) and (#**f**), we observe that the replacement has a strictly smaller size than the replaced constraint. This is also the case for each of the new replacements in rules ($\approx_\alpha$**tuple**) and (#**tuple**). Now, for the case ($\approx_\alpha$[**b**]) we have, $\tau([a]s \approx_\alpha [b]t, Pr) \gg \tau((a\ b)\cdot s \approx_\alpha t, b\#s, Pr)$ since $(|[a]s|+|[b]t|) > (|(a\ b)\cdot s|+|t|)$ such that $\tau([a]s \approx_\alpha [b]t) \gg \tau((a\ b)\cdot s \approx_\alpha t)$ and also $(|[a]s|+|[b]t|) > (|s|)$ and $(\approx_\alpha > \#)$ such that $\tau([a]s \approx_\alpha [b]t) \gg \tau(b\#s)$. For the case ($\approx_\alpha$**X**), we obtain $\tau(\phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot X, Pr) \gg \tau(\mathtt{DNF}(\bigwedge_{a\in(\mathscr{D}omP(\phi,\phi')\cup\mathscr{S}upportP(\pi,\pi'))} (\phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)) \vee a\#X), Pr))$ since, for every $a \in (\mathscr{D}omP(\phi,\phi') \cup \mathscr{S}upportP(\pi,\pi'))$, $(|\phi\hat{}\pi\cdot X|+|\phi'\hat{}\pi'\cdot X|) > (|\phi(\pi(a))|+|\phi'(\pi'(a))|)$ such that $\tau(\phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot X, Pr) \gg \tau(\phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)))$ and also $(|\phi\hat{}\pi\cdot X|+|\phi'\hat{}\pi'\cdot X|) > (|X|)$ and $(\approx_\alpha > \#)$ such that $\tau(\phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot X, Pr) \gg \tau(a\#X)$. Finally, for the case (#**X**) we observe that, $\tau(a\#\phi\hat{}\pi\cdot X, Pr) > \tau(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} (a\#\phi(b) \vee \pi^{-1}(b)\#X), Pr))$ since,

for every $(b \in (\mathscr{D}om(\phi) \cup \{a\}))$, $(|\phi\hat{}\pi\cdot X|) > (|\phi(b)|)$ such that $\tau(a\#\phi\hat{}\pi\cdot X) \gg \tau(a\#\phi(b))$ and also $(|\phi\hat{}\pi\cdot X|) > (|X|)$ such that $\tau(a\#\phi\hat{}\pi\cdot X) \gg \tau(\pi^{-1}(b)\#X)$.

Therefore, we conclude stating that the value of Pr under function τ is strictly smaller after each non-deterministic application of $\Longrightarrow$ to some C_i in Pr. Then, the result follows. ■

Confluence of the relation $\Longrightarrow$ follows from the proof of termination and the core rules having no non-trivial *overlaps*.

Lemma 3.2.6. *The relation $\Longrightarrow$ is confluent so that, if there exists a pair of simplification steps $Pr \Longrightarrow^* Pr_1$ and $Pr \Longrightarrow^* Pr_2$ then there also exists Pr_3 such that $Pr_1 \Longrightarrow^* Pr_3$ and $Pr_2 \Longrightarrow^* Pr_3$.*

Proof. See Appendix A. ■

3.2.3 Normal form of a problem

As a result of Lemma 3.2.5 & Lemma 3.2.6, the reduction rules in Definition 3.2.1 define a function that maps problems, Pr, to their *unique normal form*, as we show next.

Write $\langle Pr\rangle_{\mathtt{nf}}$ for the normal form of a problem, Pr, by application of the reduction rules from Definition 3.2.1. Then, $\langle Pr\rangle_{\mathtt{nf}}$ is a *DNF of reduced freshness constraints and clashing equalities* as follows.

Lemma 3.2.7 (Representation of normal forms)**.**

- *Each disjunct in $\langle a\#s\rangle_{\mathtt{nf}}$ consists of a conjunctive clause of reduced freshness constraints;*
- *Each disjunct in $\langle s \approx_\alpha t\rangle_{\mathtt{nf}}$ consists of a conjunctive clause of reduced freshness constraints and (possibly none) clashing equalities.*
- *An immediate consequence is $\langle Pr\rangle_{\mathtt{nf}}$ is also a DNF of reduced freshness constraints and clashing equalities when Pr is a combination of freshness and α-equality constraints.*

Proof. The result follows by application of the reduction rules from Definition 3.2.1 to non-reduced freshness constraints and non-clashing equalities. ■

The following definition is a consequence of the characterisation of normal forms for problems and Definition 3.2.2.

Definition 3.2.8. Say a problem Pr is **inconsistent** when each disjunct in $\langle Pr\rangle_{\mathtt{nf}}$ has an inconsistent constraint of form $a\#a$ or form $\bot$ or a clashing equality as described in Definition 3.2.2. Otherwise Pr is **consistent**.

The following definition is a restricted version of Definition 3.1.1, finding the set of subformulæ with respect to logical connective $\vee$. It will be applied to normal forms of problems.

Definition 3.2.9. Write $\mathbf{S}_{nf}(F)$ for the set of formulæ of a constraint formula F as defined in Definition 3.1.1 except that symbol $*$ is replaced only by binary operator $\vee$.

The definition above leads to the following straightforward corollary.

Corollary 3.2.10. *For any problem Pr,* $\mathbf{S}_{nf}(\langle Pr\rangle_{\mathtt{nf}}) \subseteq \mathbf{S}(\langle Pr\rangle_{\mathtt{nf}})$.

Example 3.2.11.

$$\mathbf{S}_{nf}(\mathtt{a\#X}, \mathtt{f(a)} \approx_\alpha \mathtt{g(a)} \vee \mathtt{b\#X}, \mathtt{a\#a}, \mathtt{c\#X} \vee \top \vee \bot) =$$
$$\{(\mathtt{a\#X}, \mathtt{f(a)} \approx_\alpha \mathtt{g(a)} \vee \mathtt{b\#X}, \mathtt{a\#a}, \mathtt{c\#X} \vee \top \vee \bot), (\mathtt{a\#X}, \mathtt{f(a)} \approx_\alpha \mathtt{g(a)}), (\mathtt{b\#X}, \mathtt{a\#a}, \mathtt{c\#X}), (\top), (\bot)\}.$$

Next, we demonstrate correctness of the reduction relation.

3.2.4 Correctness of the constraint checking algorithm

Below, we implicitly transform the representation of a conjunctive clause of primitive constraints into a set of primitive constraints, to suit the structure of entailment specified for constraints. Then, we may denote both with the same symbol (∇ or Δ or Γ) without loss of generality.

The following lemmas are a direct consequence of the property of confluence (see Lemma 3.2.6) for the reduction rules.

Lemma 3.2.12. *Let Pr, Pr' be any two arbitrary problems. Then,*

- $\langle Pr, Pr'\rangle_{\mathtt{nf}} = \mathtt{DNF}(\langle Pr\rangle_{\mathtt{nf}}, \langle Pr'\rangle_{\mathtt{nf}})$.
- *As a Corollary, if $Pr' \in \mathbf{S}(Pr)$ then $\langle Pr'\rangle_{\mathtt{nf}} \in \mathbf{S}(\langle Pr\rangle_{\mathtt{nf}})$.*

Proof. It follows by the confluence lemma from Lemma 3.2.6, the reduction relation working element-wise on the constraints in $\langle Pr, Pr'\rangle_{\mathtt{nf}}$, the implicit properties stated in Remark 3.2.3 and the definitions of both $\mathbf{S}(\cdot)$ (see Definition 3.1.1) and DNF of a formula. ■

Below, we formalize the correspondence between the normal form of a problem and the judgement relation over constraints, $\vdash$.

Lemma 3.2.13. *Assume $Pr \Longrightarrow_* Pr'$ and ∇ a freshness context. Then*

- $\nabla \vdash Pr \Longleftrightarrow \nabla \vdash Pr'$.

- *An immediate consequence is* $\nabla \vdash Pr \Longleftrightarrow \nabla \vdash \langle Pr \rangle_{\mathtt{nf}}$.

Proof. Any reduction step $Pr \Longrightarrow Pr'$ corresponds to one of the 12 reduction rules described in Definition 3.2.1. Accordingly, each reduction rule corresponds precisely to one of the syntax-directed core derivation rules introduced in Definition 2.2.3 and Definition 2.2.5 for the freshness and α-equality entailment relation respectively. The result follows by induction on the number of reduction steps in the simplification $Pr \Longrightarrow_* Pr'$. ■

The next corollary is a consequence of the lemma above.

Corollary 3.2.14. *Assume* ∇ *is a freshness context. If* $\nabla \vdash Pr$ *then Pr is consistent.*

Proof. It follows by a simple induction on the derivations of *Pr*. ■

The lemma below shows the correspondence between consistent constraint problems and entailment of a problem.

Lemma 3.2.15. *Suppose* $\langle Pr \rangle_{\mathtt{nf}} = Cr_1 \vee \ldots \vee Cr_n$ *where each* Cr_i *is a conjunctive clause of reduced freshness constraints and (possibly none) clashing equalities, as given in Lemma 3.2.7. Then,* $\exists Cr_i \in \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}}) : Cr_i \vdash Pr \Longleftrightarrow Pr$ *is consistent, for* $1 \leq i \leq n$.

Proof. Suppose *Pr* is consistent. By Definition 3.2.8 there is some $Cr_i \in \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$ such that Cr_i is a conjunctive clause of consistent freshness constraints, that is, a freshness context (by a trivial transformation of conjunctive clause to sets). Then, we have $Cr_i \vdash \langle Pr \rangle_{\mathtt{nf}}$ trivially, since $Cr_i \in \mathbf{S}_{nf}(Cr_1 \vee \ldots \vee Cr_n)$ for some $1 \leq i \leq n$. The result follows by Lemma 3.2.13.

Conversely, if $Cr_i \in \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$ and $Cr_i \vdash Pr$ then the result follows by Corollary 3.2.14. ■

This leads to the main theorem of this section.

Theorem 3.2.16 (Correctness)**.** *Let Pr be a problem,* ∇, Δ *a pair of freshness contexts. Then,*

- *if* $\Delta \subseteq \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$ *then* $\Delta \vdash Pr$ *(Soundness);*
- $\exists \Delta \subseteq \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$ *such that, if* $\nabla \vdash Pr$ *then* $\nabla \vdash \Delta$ *(completeness).*

Proof. For the first claim.

Suppose $\Delta \subseteq \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$. Then, by Lemma 3.2.15 (and a trivial transformation to sets) we have, $\Delta \vdash \langle Pr \rangle_{\mathtt{nf}}$ and by Lemma 3.2.13 $\Delta \vdash Pr$.

For the second claim.

Suppose $\nabla \vdash Pr$. By the assumptions and Corollary 3.2.14, *Pr* is consistent. By Definition 3.2.8 and Lemma 3.2.7, $\exists \Delta \subseteq \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$ such that Δ is consistent. by Lemma 3.2.13 $\nabla \vdash \Delta$ for some $\Delta \in \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$. ■

Corollary 3.2.17 (Cut rule)**.** *Let* ∇, Δ *be a pair of freshness contexts and* Pr, Pr' *a pair of problems.*

- *If* $\Delta \vdash \nabla$ *and* $\Delta, \nabla \vdash \Gamma$*, then* $\Delta \vdash \Gamma$.
- *In particular, if* $\Delta \vdash Pr$ *and* $\Delta, \nabla_i \vdash Pr'$ *for some* $\nabla_i \subseteq \mathbf{S}_{nf}(\langle Pr \rangle_{\mathtt{nf}})$ *such that* $\Delta \vdash \nabla_i$ *then,* $\Delta \vdash Pr'$.

Proof. For the general case.

Suppose $\Delta \vdash \nabla$. Then, by the completeness claim from Theorem 3.2.16, $\Delta \vdash C$ for each $C \in \nabla$. Therefore we can convert a derivation of $\Delta, \nabla \vdash \Gamma$ into a derivation of $\Delta \vdash \Gamma$ as required.

For the particular case.

It follows by the previous case and Theorem 3.2.16. Following the notation above, notice the necessary condition $\Delta \vdash \nabla_i$ for the particular case to hold due to the possibility of disjuncts resulting from the derivation of problem Pr. ■

We spend the rest of the chapter refining the derivability process to facilitate the automatic removal of superfluous results from the normal form of a problem.

3.3 Discarding Redundant Disjuncts

Unlike standard nominal terms (see Theorem 17 in (Fernández and Gabbay, 2007)), the normal form of a problem Pr, $\langle Pr \rangle_{\mathtt{nf}}$, as stated in Lemma 3.2.7 may contain not only disjuncts providing evidence for the entailment relation to hold but also disjuncts which have inconsistent constraints or clashing equalities, possibly both. This is due to the disjunctive nature of the premises in core rules (#**X**) and ($\approx_\alpha$**X**) and the absence of a freshness context to provide evidence of derivability when variables are encountered during the reduction path. By removing these superfluous subformulæ during the simplification process of Pr, what it remains is either a DNF holding distinct evidence required for a derivation proof or, otherwise, a proof of Pr being inconsistent.

In this section, we introduce an enhancement to the derivability algorithm in order to filter these superfluous results out during the application of core rules (#**X**) and ($\approx_\alpha$**X**) to a problem.

In the sequel, we need to make a distinction between the derivability algorithm from Definition 3.2.1 and its augmented version. To that extent, we refer to the derivability algorithm we present later, in Definition 3.3.6, as **enhanced algorithm**, to avoid confusion.

We begin by specifying the notion of redundant disjunct and providing a solution to discard redundant disjuncts from the normal form of a problem. The section then continues by providing some examples and a proof of correctness for the enhanced algorithm.

3.3.1 An auxiliary function to discard redundant disjuncts

Application of core rules $(\approx_\alpha \mathbf{X})$ and $(\#\mathbf{X})$ given in Definition 3.2.1 produces a DNF of reduced freshness constraints and clashing equalities where each consistent disjunct entails a successful derivation of the problem. However, the logic is too broad and some disjuncts are redundant, either because they contain inconsistent constraints or clashing equalities or because a derivation proof can be obtained without evidence from a freshness context, that is, the formula is consistent albeit redundant. We illustrate the notion of redundant disjuncts with a pair of examples.

Example 3.3.1. Using the set of simplification rules in Definition 3.2.1,
$\mathtt{C} = [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X} \approx_\alpha [\mathtt{a} \mapsto \mathtt{f}(\mathtt{b})]\cdot\mathtt{X}$ is reduced as follows.

$$[\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X} \approx_\alpha [\mathtt{a} \mapsto \mathtt{f}(\mathtt{b})]\cdot\mathtt{X} \overset{(\approx_\alpha \mathbf{X})}{\Longrightarrow} \mathtt{a}\#\mathtt{X} \vee \mathtt{b} \approx_\alpha \mathtt{f}(\mathtt{b}).$$

Hence, $\langle C \rangle_{\mathtt{nf}} = \mathtt{a}\#\mathtt{X} \vee \mathtt{b} \approx_\alpha \mathtt{f}(\mathtt{b})$, however, only $\mathtt{a}\#\mathtt{X}$ entails the derivation of C, $\mathtt{a}\#\mathtt{X} \vdash C$ whereas $\mathtt{b} \approx_\alpha \mathtt{f}(\mathtt{b})$ is a clashing equality and thus a redundant result.

Example 3.3.2. Using the set of simplification rules in Definition 3.2.1, $s = \mathtt{a}\#[\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X}$ is reduced as follows.

$$\mathtt{a}\#[\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X} \overset{(\#\mathbf{X})}{\Longrightarrow} \mathtt{a}\#\mathtt{b} \vee \mathtt{a}\#\mathtt{X} \overset{(\#\mathbf{ab})}{\Longrightarrow} \top \vee \mathtt{a}\#\mathtt{X}.$$

Hence, $\langle s \rangle_{\mathtt{nf}} = \top \vee \mathtt{a}\#\mathtt{X}$. However, $\mathtt{a}\#\mathtt{X}$ is redundant in providing satisfiability of the freshness constraint derivation; any instantiation of $\mathtt{X}$ by a term containing unabstracted occurrences of atom $\mathtt{a}$ is under the action of a-substitution $[\mathtt{a} \mapsto \mathtt{b}]$. Therefore, $\mathtt{a}\#\mathtt{X}$ is consistent yet redundant.

Note that the same issue occurred back in Example 3.2.4 where the normal form of $[\mathtt{a}][\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X} \approx_\alpha [\mathtt{b}][\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X}$ was the DNF $\top \vee \mathtt{b}\#\mathtt{X} \vee \mathtt{a}\#\mathtt{X} \vee (\mathtt{a}\#\mathtt{X}, \mathtt{b}\#\mathtt{X})$. However, if the α-equality relation is derivable from an empty freshness context, denoted by $\top$, any other entailment is valid yet unnecessary, weakening the judgement.

Definition 3.3.3 (Redundant disjunct)**.** Given a disjunctive clause of constraints $C \vee C'$ generated during the application of core rules $(\approx_\alpha \mathbf{X})$ and $(\#\mathbf{X})$, we refer to any disjunct C as a **redundant disjunct** if it contains inconsistent constraints, clashing equalities or if disjunct

$C' = \top$. The definition extends naturally to the disjuncts occurring in the normal form of a problem Pr, $\langle Pr \rangle_{\mathtt{nf}}$.

It is of interest to discard redundant disjuncts from a normal form to a problem Pr, reducing the formula to a DNF of consistent results entailing the derivation of the problem or providing instead a unique result to represent failure of the derivability process, denoted here by $\bot$.

In order to remove redundant disjuncts during the reduction relation, an auxiliary function is added on the RHS of core rules (#**X**) and ($\approx_\alpha$**X**) to control the formation of reduction paths leading to superfluous or inconsistent results. We have seen this notion before, in the definition of both `fresh` and `ds`, as stated in the premises of Definition 2.2.3 (see Equation 2.2) and Definition 2.2.5 (see Equation 2.4) respectively. However, both definitions were parametrised by a freshness context, thus ruling non-determinism, in the form of a disjunction, out of the process. The 'helper' function $\langle \cdot, \cdot, \cdot \rangle$ included on the RHS of both variable cases filters out redundant disjuncts by finding their normal form. This is formalise next.

Definition 3.3.4 (Auxiliary function $\langle a, C, X \rangle$)**.** For any $a, b \in \mathscr{A}, X \in \mathscr{X}$, terms s, t and constraint C such that $C = b\#s$ or $C = s \approx_\alpha t$, the function $\langle \cdot, \cdot, \cdot \rangle$ is defined over a freshness judgement as follows

$$\langle a, C, X \rangle = \begin{cases} \top & \text{if} \quad \vdash C \quad (1) \\ a\#X & \text{if} \quad \nexists \nabla. \nabla \vdash C \quad (2) \\ \langle C \rangle_{\mathtt{nf}} \vee a\#X & \text{if} \quad \exists \nabla. \nabla \vdash C \quad (3) \end{cases}$$

where $\langle \cdot \rangle_{\mathtt{nf}}$ is the revised normal form of a problem as introduced in Lemma 3.3.8.

Following the notation above we observe that, a successful derivation proof of relation C from an empty freshness context, shown in (1), implies that no further checks must be carried out and constraint $\top$ is returned. On the other hand, if C has no solution, as noted in (2), C is removed from the reduction relation and a primitive constraint of form $a\#X$ is produced to prevent the derivation of C by some instantiation of X containing atom a unabstracted. In Definition 3.3.4, parameter a stands for:

- Case ($C_{\approx_\alpha}$). Any atom occurring in the domain of an a-substitution or in the support of a permutation suspended at either side of predicate $\approx_\alpha$. Then, $C = \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))$ for a pair of atom actions $\phi \hat{} \pi$, $\phi' \hat{} \pi'$ of X;

- Case ($C_\#$). Any atom in the domain of an a-substitution ϕ suspended on X and atom b where $C = b\#\phi(a)$.

The last condition, (3), preserves both reduction paths, that is, the normal form of constraint C and the primitive constraint that prevents the formation of constraint C by instantiation of X, $a\#X$. Note that this case is produced only when suspended variables are encountered during the reduction of C since further evidence for the suspended variable is required for the derivability proof to hold (therefore $\langle C \rangle_{\mathtt{nf}} \neq \top$ as in case (1)).

Looking back at Example 3.3.1 and Example 3.3.2, we now observe that the result is reduced to $\mathtt{a\#X}$ via case (2) and $\top$ via case (1) respectively, if applying the core rules augmented with function $\langle \cdot,\cdot,\cdot \rangle$, namely, $\langle a, b, X \approx_\alpha f(b) \rangle$ and $\langle a, a\#b, X \rangle$ respectively. Here is a further example.

Example 3.3.5. Using DNF and Definition 3.3.4 on $[\mathtt{a} \mapsto \mathtt{Y}]\hat{}(\mathtt{a\ b})\cdot\mathtt{X} \approx_\alpha [\mathtt{b} \mapsto (\mathtt{a\ b})\cdot\mathtt{Y}]\cdot\mathtt{X}$, we obtain

$$\mathtt{DNF}(\langle \mathtt{a}, \mathtt{b}, \mathtt{X} \approx_\alpha \mathtt{a} \rangle \wedge \langle \mathtt{b}, \mathtt{Y} \approx_\alpha (\mathtt{a\ b})\cdot\mathtt{Y}, \mathtt{X} \rangle) = \mathtt{DNF}(\mathtt{a\#X} \wedge ((\mathtt{a\#Y}, \mathtt{b\#Y}) \vee \mathtt{b\#X})) =$$
$$\mathtt{a\#X}, \mathtt{a\#Y}, \mathtt{b\#Y} \vee \mathtt{a\#X}, \mathtt{b\#X}.$$

The recursive call of function $\langle \cdot,\cdot,\cdot \rangle$ on the atom actions will be handled directly by the enhanced variable reduction rule as we show in Definition 3.3.6.

Next, the auxiliary functions discussed in this section are added to the constraint checking algorithm along with a pair of rules to deal with inconsistent freshness contexts.

3.3.2 An enhanced constraint checking algorithm

The set of core rules in the derivability algorithm from Definition 3.2.1 is now extended by adding a new pair of reduction rules to deal with the possibility of failure, namely (#⊥) and rule ($\approx_\alpha\bot$). In addition, variable rules ($\approx_\alpha\mathbf{X}$) and (#**X**) are augmented to filter redundant disjuncts. These adjustments to the constraint checking algorithm have been added to the enhanced algorithm below, in Definition 3.3.6.

The additional reduction rule (#⊥) transforms any problem containing a constraint of form $a\#a$ into $\bot$. Similarly, rule ($\approx_\alpha\bot$) reduces to $\bot$ any problem containing a clashing α-equality. As a result, rules (#⊥) and ($\approx_\alpha\bot$) reduce the number of computations when dealing with failure, portraying $\bot$ as the result of the reduction relation.

The enhanced algorithm is described as follows.

Definition 3.3.6 (Enhanced algorithm)**.** Let Pr be a constraint problem as in Definition 3.1.3, a, b a pair of distinct atoms and $\phi\hat{}\pi\cdot X, \phi'\hat{}\pi'\cdot X$ a pair of occurrences of variable X In addition, $\langle \cdot,\cdot,\cdot \rangle$ is the function described in Definition 3.3.4.

Then, $\Longrightarrow$ is the reduction relation defined by the set of core rules from Definition 3.2.1 with rules (#**X**) and ($\approx_\alpha$**X**) replaced by rules (#$_\mathbf{e}$**X**) and ($\approx_{\alpha_\mathbf{e}}$**X**) respectively, and extended with rules (#$\bot$), ($\approx_\alpha\bot$) and also (#$\varnothing$).

Additional rules are defined below.

$$(\approx_{\alpha_\mathbf{e}}\mathbf{X})\quad \phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot X,\ Pr \implies \mathtt{DNF}\Big(\bigwedge_{a\in(\mathscr{D}omP(\phi,\phi')\cup\mathscr{S}upportP(\pi,\pi'))} \langle\, a, \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)), X\,\rangle, Pr\Big)$$

$$(\approx_\alpha\bot)\quad s\approx_\alpha t,\ Pr \implies \bot \quad (\text{ where } s\approx_\alpha t \text{ is clashing})$$

$$(\#\bot)\quad a\#a,\ Pr \implies \bot$$

$$(\#_\mathbf{e}\mathbf{X})\quad a\#\phi\hat{}\pi\cdot X,\ Pr \implies \mathtt{DNF}\Big(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X\,\rangle, Pr\Big) \quad (\phi\neq\mathtt{Id}\vee\pi\neq\mathtt{Id})$$

A clarifying example follows.

Example 3.3.7. Taking as motivation the differentiation and λ-calculus signature from Example 2.1.1 and Example 1.2.1 respectively, we introduce a pair of a freshness and an α-equality constraint, showing below their derivation by application of the derivability algorithm from Definition 3.3.6.

• $Pr = \{\mathtt{a\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X))}, \mathtt{b\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X))},$
$\mathtt{app(abs([c]M),N)} \approx_\alpha \mathtt{app(abs([d]M),N)}\}$ resolves to
$\mathtt{b\#Y, b\#F, a\#F, c\#M, d\#M, a\#X, b\#G, b\#X}$
$\vee\ \mathtt{a\#F, b\#F, c\#M, d\#M, a\#X, b\#G, b\#X}$
$\vee\mathtt{a\#F, b\#F, a\#Y, c\#M, d\#M, a\#X, b\#G, b\#X}$
$\vee\mathtt{b\#Y, b\#F, a\#Y, c\#M, d\#M, a\#X, b\#G, b\#X}$ as follows:

- $\mathtt{a\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)), b\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)),}$
  $\mathtt{app(abs([c]M),N)} \approx_\alpha \mathtt{app(abs([d]M),N)}$
  $\overset{(\approx_\alpha\mathbf{f})}{\Longrightarrow} \mathtt{a\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)), b\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)),}$
  $\mathtt{(abs([c]M),N)} \approx_\alpha \mathtt{(abs([d]M),N)}$
  $\overset{(\approx_\alpha\mathbf{tupl})}{\Longrightarrow} \mathtt{a\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)), b\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)),}$
  $\mathtt{abs([c]M)} \approx_\alpha \mathtt{abs([d]M), N} \approx_\alpha \mathtt{N}$
  $\overset{(\approx_\alpha\mathbf{f})}{\Longrightarrow} \mathtt{a\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)), b\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)),}$
  $\mathtt{([c]M)} \approx_\alpha \mathtt{([d]M), N} \approx_\alpha \mathtt{N}$
  $\overset{(\approx_\alpha\mathbf{tupl})}{\Longrightarrow} \mathtt{a\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)), b\#(cos[a\mapsto Y]\cdot F \times diff([a]G,X)),}$

$[\mathtt{c}]\mathtt{M} \approx_\alpha [\mathtt{d}]\mathtt{M}, \mathtt{N} \approx_\alpha \mathtt{N}$
$\overset{(\approx_{\alpha_{\mathbf{e}}}\mathbf{X})}{\Longrightarrow} \mathtt{DNF}(\mathtt{a\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})),$
$[\mathtt{c}]\mathtt{M} \approx_\alpha [\mathtt{d}]\mathtt{M})$
$\overset{(\approx_{\alpha}[\mathbf{b}])}{\Longrightarrow} \mathtt{a\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})),$
$(\mathtt{d}\ \mathtt{c})\cdot\mathtt{M} \approx_\alpha \mathtt{M}, \mathtt{d\#M}$
$\overset{(\approx_{\alpha_{\mathbf{e}}}\mathbf{X})}{\Longrightarrow} \mathtt{DNF}(\langle\, \mathtt{c}, \mathtt{d} \approx_\alpha \mathtt{c}, \mathtt{M} \,\rangle, \langle\, \mathtt{d}, \mathtt{c} \approx_\alpha \mathtt{d}, \mathtt{M} \,\rangle, \mathtt{a\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})),$
$\mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{d\#M})$
$= \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{d\#M}$
$\overset{(\mathbf{\#tupl})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#diff}([\mathtt{a}]\mathtt{G}, \mathtt{X}), \mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{d\#M}$
$\overset{(\mathbf{\#f})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#diff}([\mathtt{a}]\mathtt{G}, \mathtt{X}), \mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{d\#M}$
$\overset{(\mathbf{\#f})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#}([\mathtt{a}]\mathtt{G}, \mathtt{X}), \mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{d\#M}$
$\overset{(\mathbf{\#tupl})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#}[\mathtt{a}]\mathtt{G}, \mathtt{a\#X}, \mathtt{b\#}(\mathtt{cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F} \times \mathtt{diff}([\mathtt{a}]\mathtt{G}, \mathtt{X})), \mathtt{d\#M}$
$\overset{(\mathbf{\#tupl})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#}[\mathtt{a}]\mathtt{G}, \mathtt{a\#X}, \mathtt{b\#cos}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#diff}([\mathtt{a}]\mathtt{G}, \mathtt{X}), \mathtt{d\#M}$
$\overset{(\mathbf{\#f})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#}[\mathtt{a}]\mathtt{G}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#diff}([\mathtt{a}]\mathtt{G}, \mathtt{X}), \mathtt{d\#M}$
$\overset{(\mathbf{\#f})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#}[\mathtt{a}]\mathtt{G}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#}([\mathtt{a}]\mathtt{G}, \mathtt{X}), \mathtt{d\#M}$
$\overset{(\mathbf{\#tupl})}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#}[\mathtt{a}]\mathtt{G}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#}[\mathtt{a}]\mathtt{G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\overset{(\mathbf{\#}[\mathbf{a}])}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#}[\mathtt{a}]\mathtt{G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\overset{(\mathbf{\#}[\mathbf{b}])}{\Longrightarrow} \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\overset{(\mathbf{\#_e X})}{\Longrightarrow} \mathtt{DNF}(\langle\, \mathtt{a}, \mathtt{a\#Y}, \mathtt{F} \,\rangle, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M})$
$= \mathtt{DNF}((\mathtt{a\#F} \vee \mathtt{a\#Y}), \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M})$
$= \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\overset{(\mathbf{\#_e X})}{\Longrightarrow} \mathtt{DNF}(\langle\, \mathtt{a}, \mathtt{b\#Y}, \mathtt{F} \,\rangle, \langle\, \mathtt{b}, \mathtt{b\#b}, \mathtt{F} \,\rangle, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M})$
$\vee \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$= \mathtt{DNF}((\mathtt{a\#F} \vee \mathtt{b\#Y}), \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M})$
$\vee \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$= \mathtt{b\#Y}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee\ \mathtt{a\#F}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#}[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{F}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\overset{(\mathbf{\#_e X})}{\Longrightarrow} \mathtt{b\#Y}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee\ \mathtt{a\#F}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee \mathtt{DNF}(\langle\, \mathtt{a}, \mathtt{b\#Y}, \mathtt{F} \,\rangle, \langle\, \mathtt{b}, \mathtt{b\#b}, \mathtt{F} \,\rangle, \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M})$
$= \mathtt{b\#Y}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee\ \mathtt{a\#F}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$

$\vee \mathtt{DNF}\big((\mathtt{a\#F} \vee \mathtt{b\#Y}), \mathtt{b\#F}, \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}\big)$
$= \mathtt{b\#Y}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee\ \mathtt{a\#F}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee \mathtt{a\#F}, \mathtt{b\#F}, \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$\vee \mathtt{b\#Y}, \mathtt{b\#F}, \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}, \mathtt{d\#M}$
$= \mathtt{b\#Y}, \mathtt{b\#F}, \mathtt{a\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}$
$\vee\ \mathtt{a\#F}, \mathtt{b\#F}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}$
$\vee \mathtt{a\#F}, \mathtt{b\#F}, \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}$
$\vee \mathtt{b\#Y}, \mathtt{b\#F}, \mathtt{a\#Y}, \mathtt{c\#M}, \mathtt{d\#M}, \mathtt{a\#X}, \mathtt{b\#G}, \mathtt{b\#X}$

Next, Lemma 3.2.7 is updated to reflect the modifications added to the core rules from Definition 3.2.1.

Lemma 3.3.8 (Normal forms update)**.** *Both* $\langle a\#s\rangle_{\mathtt{nf}}, \langle s \approx_\alpha t\rangle_{\mathtt{nf}}$ *are, either a DNF of consistent freshness constraints or* $\bot$*.*

As a corollary, $\langle Pr\rangle_{\mathtt{nf}}$ *is also of the same form as above, for any problem Pr.*

Proof. By induction on the derivation of both relations $a\#s$ and $s \approx_\alpha t$ using the set of rules given in Definition 3.3.6.

Previously, in Lemma 3.2.7, it was proved that both $\langle a\#s\rangle_{\mathtt{nf}}$ and $\langle s \approx_\alpha t\rangle_{\mathtt{nf}}$ consist of a DNF of reduced freshness constraints the former and a DNF of both reduced freshness constraints and (possibly none) clashing equalities the latter. Now, we need to show that the normal form of any freshness or α-equality constraint is either a DNF of consistent freshness constraints or, otherwise, the constraint for derivation failure, $\bot$. This is the case, as a result of auxiliary function $\langle \cdot, \cdot, \cdot \rangle$ (see Definition 3.3.4), added to the RHS of rules $(\#_{\mathbf{e}}\mathbf{X})$ and $(\approx_{\alpha_{\mathbf{e}}}\mathbf{X})$, which filters out any redundant disjunct and failure rules $(\approx_\alpha \bot), (\approx_\alpha \#)$ added to the enhanced algorithm, which reduce any conjunctive clause containing inconsistent constraints or clashing equalities into $\bot$. Finally, the result follows by application of Remark 3.2.3. ■

The next lemma shows the correspondence between variable cases $(\approx_{\alpha_{\mathbf{e}}}\mathbf{X}), (\#_{\mathbf{e}}\mathbf{X})$ from the enhanced algorithm in Definition 3.3.6 and variable cases $(\approx_\alpha \mathbf{X}), (\#\mathbf{X})$ from the derivability algorithm given in Definition 3.2.1. Basically, the normal form of the formula on the RHS of rule $(\approx_{\alpha_{\mathbf{e}}}\mathbf{X})$ and rule $(\#_{\mathbf{e}}\mathbf{X})$ occurs in the set of subformulæ of the normal form of the formula on the RHS of rule $(\approx_\alpha \mathbf{X})$ and rule $(\#\mathbf{X})$ respectively, since the addition of failure rules $(\approx_\alpha \bot), (\#\bot)$ and the auxiliary function from Definition 3.3.4 does not add constraints to the result but discards redundant disjuncts from occurring in the normal form of a variable constraint.

Lemma 3.3.9. *Let $\phi\hat{}\pi{\cdot}X, \phi'\hat{}\pi'{\cdot}X$ be a pair of occurrences of variable $X \in \mathscr{X}$ and a an atom.*

Suppose $\langle \mathtt{DNF}(\bigwedge_{a\in(\mathscr{D}omP(\phi,\phi')\cup\mathscr{S}upportP(\pi,\pi'))} (\phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)) \vee a\#X))\rangle_{\mathtt{nf}}$
(resp. $\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} (a\#\phi(b\langle\, \pi^{-1}(b), a\#\phi(b), X\,\rangle) \vee \pi^{-1}(b)\#X)))$ *is the DNF formula obtained by application of $\langle\cdot\rangle_{\mathtt{nf}}$ from Lemma 3.2.7 and rule $(\approx_\alpha \mathbf{X_c})$ (resp. rule $(\#\mathbf{X_c})$) from Definition 3.2.1 to $\phi\hat{}\pi{\cdot}X \approx_\alpha \phi'\hat{}\pi'{\cdot}X$ (resp. $a\#\phi\hat{}\pi{\cdot}X$).*

Suppose also that $\langle \mathtt{DNF}(\bigwedge_{a\in(\mathscr{D}omP(\phi,\phi')\cup\mathscr{S}upportP(\pi,\pi'))} \langle\, a, \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)), X\,\rangle)\rangle_{\mathtt{nf}}$
(resp. $\langle \mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X\,\rangle)\rangle_{\mathtt{nf}}$*) is the DNF formula obtained by application of $\langle\cdot\rangle_{\mathtt{nf}}$ from Lemma 3.3.8 and rule $(\approx_\alpha \mathbf{X_e})$ (resp. rule $(\approx_\alpha \mathbf{X_e})$) from Definition 3.3.6 to $\phi\hat{}\pi{\cdot}X \approx_\alpha \phi'\hat{}\pi'{\cdot}X$ (resp. $a\#\phi\hat{}\pi{\cdot}X$). Then,*

1. $\langle \mathtt{DNF}(\bigwedge_{a\in(\mathscr{D}omP(\phi,\phi')\cup\mathscr{S}upportP(\pi,\pi'))} \langle\, a, \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)), X\,\rangle)\rangle_{\mathtt{nf}}$
   $\in \mathbf{S}(\langle \mathtt{DNF}(\bigwedge_{a\in(\mathscr{D}omP(\phi,\phi')\cup\mathscr{S}upportP(\pi,\pi'))} (\phi(\pi(a)) \approx_\alpha \phi'(\pi'(a)) \vee a\#X))\rangle_{\mathtt{nf}})$;

2. $\langle \mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X\,\rangle)\rangle_{\mathtt{nf}}$
   $\in \mathbf{S}(\langle \mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} (a\#\phi(b) \vee \pi^{-1}(b)\#X))\rangle_{\mathtt{nf}})$.

Proof. Both claims follow from rules $(\#\bot), (\approx_\alpha \bot)$ converting inconsistent constraint and clashing equalities into constraint $\bot$, rules $(\approx_{\alpha_e}\mathbf{X})$ and $(\#_e\mathbf{X})$ being similar to rules $(\approx_\alpha \mathbf{X})$ and $(\#\mathbf{X})$ respectively, except that rules $(\approx_{\alpha_e}\mathbf{X})$ and $(\#_e\mathbf{X})$ include the auxiliary function from Definition 3.3.4 on their RHS formula, discarding the formation of redundant disjuncts and also by the corollary stated in the second claim of Lemma 3.2.12. ■

3.4 Conclusion

This chapter introduced a pair of constraint checking algorithms for extended nominal terms following the set of core freshness and α-equality inference rules given in Chapter 2.

The first algorithm presented in Definition 3.2.1 generates by exhaustive search a DNF of reduced freshness constraints and (possibly none) clashing equalities as a solution to a constraint problem. Its correctness was demonstrated in Theorem 3.2.16 and the normal form characterised in Lemma 3.2.7.

The second algorithm given in Definition 3.3.6 is a variation on the previous one where redundant results are discarded during the execution of the algorithm in order to provide a normal form for a problem containing only freshness contexts which entail the derivation of

the given problem or $\perp$ otherwise. The normal form of the algorithm was then updated and proven in Lemma 3.3.8.

Now, we have set the foundation from where to build a unification procedure, as we show in the following chapters.

Part III

Unification and Matching of Extended Terms

Chapter 4

Nominal Unification in the Presence of Atom Substitutions

We now turn our attention to *unifiability* in eNRSs.

Generally speaking, unifiability of terms s,t, denoted here by $s \stackrel{?}{\approx}_? t$, is concerned with replacing variables in either s or t such that both terms are logically equal, that is, there exists a v-substitution σ such that $s\sigma \approx_\alpha t\sigma$ by means of predicate $\approx_\alpha$ introduced previously in Chapter 2. In nominal theory, instances of $\approx_\alpha$ are derived using #. Hence, nominal unification is concerned with whether there exists a v-substitution σ and a freshness context ∇ for which $\nabla \vdash s\sigma \approx_\alpha t\sigma$ holds. Then, s and t are said to be *unifiable*.

Additionally, in the extended nominal framework, suspended a-substitutions increase the complexity of the unification process by introducing non-determinism of freshness results for a given unifier, non-uniqueness of *most general unifiers* and the selection of *fresher* atoms when acting on instances of variables (see Definition 2.3.2). In fact, it is the aim of this chapter to demonstrate that, similarly to higher-order unification of lambda-terms, unification of nominal terms extended with a-substitution is indeed undecidable. The proof of undecidability is adapted from the seminal paper (Goldfarb, 1981) on undecidability of second-order unification. This is done in Section 4.3. Previous to that, we spend the chapter providing a notion of nominal unification suitable for the complexities of our extended framework.

4.1 Unification Problems and their Solutions

4.1.1 Structure of a unification problem

Definition 4.1.1 (Unification problem)**.** A **unification problem** Pr over a signature is a set of constraints as previously defined in Definition 3.1.3 where equality constraints $s \approx_\alpha t$ are replaced by **unification constraints** $s \;{}_?{\approx}_?\; t$.

Solving a unification constraint, $s \;{}_?{\approx}_?\; t$, also involves the derivation of freshness constraints as a result of simplification rules $({}_?{\approx}_?\mathbf{X})$ and $({}_?{\approx}_?[\mathbf{b}])$ (previously $(\approx_\alpha \mathbf{X})$ and $(\approx_\alpha [\mathbf{b}])$ respectively). Furthermore, freshness constraints are not only added during the unification process, but could also be chosen arbitrarily, as restrictions on names of atoms occurring during variable instantiation. For that matter, primitive constraints that are part of the definition of a term (i.e., a term-in-context) are also taken into account in the unification process. This topic becomes more relevant in the following chapter, when dealing with nominal rewriting and the matching of terms.

Next, we extend the definition of unification problem to include pairs of terms-in-context. The definition deals only with sets of consistent freshness contexts (falsity, $\bot$, as an inconsistent set would entail every freshness and α-equality constraint C).

Definition 4.1.2 (Unification problem-in-context)**.** A **unification problem-in-context** consists of a pair of terms-in-contexts $\nabla \vdash s$ and $\Delta \vdash t$ over predicate ${}_?{\approx}_?$. It is denoted by $(\nabla \vdash s) \;{}_?{\approx}_?\; (\Delta \vdash t)$ and defined as $\{s \;{}_?{\approx}_?\; t\} \cup \nabla \cup \Delta$.

Intuitively, a unification problem-in-context is just a unification problem as given in Definition 4.1.1 containing a single unification constraint of form $s \;{}_?{\approx}_?\; t$ and additional freshness conditions $\nabla \cup \Delta$ which must be satisfied by any v-substitution σ which unifies s and t, that is, $\Gamma \vdash \nabla\sigma, \Delta\sigma$ for some freshness context Γ.

4.1.2 α-equality normal form for unification problems

In chapter 3, Lemma 3.3.8 we demonstrated that both $\langle s \approx_\alpha t \rangle_{\mathtt{nf}}$ and $\langle a\#s \rangle_{\mathtt{nf}}$ consist of either a DNF of consistent freshness constraints or $\bot$. In this chapter, the set of clashing equalities described in Definition 3.2.2 is revised to reflect the representation of unification constraints, namely, any clashing equality $\phi\hat{}\pi{\cdot}X \approx_\alpha t$ (resp. $t \approx_\alpha \phi\hat{}\pi{\cdot}X$) where the disagreement is between a moderated variable, $\phi\hat{}\pi{\cdot}X$ for some $X \in \mathscr{X}$ and any other extended nominal structure t must be recovered and preserved as an α-equivalence normal form. Later, these recovered clashing equalities will generate the v-substitutions required for the solving of unification constraints. Further, we need to represent the DNF formula $\langle Pr \rangle_{\mathtt{nf}}$, where Pr is a

derivability problem as described in 3.1.3, as a collection of sets, to preserve the same data structure across the chapter. The notion of a normal form is updated below.

Lemma 4.1.3 (α-equality normal form for the unification theory)**.**

- $\langle a\#s\rangle_{\mathtt{nf}}$ *is a collection of sets,* $\{\nabla_i \mid i \in I\}$*, where each* ∇_i *is a freshness context or otherwise,* $\langle a\#s\rangle_{\mathtt{nf}} = \{\bot\}$.

- $\langle s \approx_\alpha t\rangle_{\mathtt{nf}}$ *is a collection of sets,* $\{\nabla_i \mid i \in I\}$*, where each* ∇_i *consists of primitive constraints and (possibly none) clashing equalities of form* $\phi\hat{}\pi{\cdot}X \approx_\alpha t$ *or form* $t \approx_\alpha \phi\hat{}\pi{\cdot}X$ *for some* $X \in \mathscr{X}$ *where* t *is any term with a root symbol distinct to* X *or otherwise,* $\langle s \approx_\alpha t\rangle_{\mathtt{nf}} = \{\bot\}$.

- *An immediate consequence is that* $\langle Pr\rangle_{\mathtt{nf}}$ *is also a collection of sets of form* $\langle s \approx_\alpha t\rangle_{\mathtt{nf}}$ *as above or otherwise,* $\langle Pr\rangle_{\mathtt{nf}} = \bot$.

Proof. It follows from Lemma 3.3.8 and the modification to the set of clashing equalities (see Definition 3.2.2). There is a trivial transformation of any derivability problem Pr from set to formula before application of function $\langle\cdot\rangle_{\mathtt{nf}}$ to Pr. The rest of the proof can be found in Appendix B. ■

We drop the outer brackets for singleton $\{\varnothing\}$ and $\{\bot\}$, that is, $\langle Pr\rangle_{\mathtt{nf}} = \varnothing$ and $\langle Pr\rangle_{\mathtt{nf}} = \bot$.

Sometimes we abuse the notation of $\langle\cdot\rangle_{\mathtt{nf}}$ and apply it to unification problems, Pr. Then, $\langle Pr\rangle_{\mathtt{nf}}$ is interpreted as the normal form of Pr, where each unification predicate $_?\!\approx\!_?$ has been replaced by predicate $\approx_\alpha$ before execution of function $\langle\cdot\rangle_{\mathtt{nf}}$ and replaced back to predicate $_?\!\approx\!_?$ when displaying its result.

In the sequel, clashing equalities refer to the set of constraints enumerated in Definition 3.2.2 minus clashing equalities of form $\phi\hat{}\pi{\cdot}X \approx_\alpha t$ (resp. $t \approx_\alpha \phi\hat{}\pi{\cdot}X$) where the disagreement is between a moderated variable, $\phi\hat{}\pi{\cdot}X$ for some $X \in \mathscr{X}$ and any other extended nominal structure t. Additionally, $\langle\cdot\rangle_{\mathtt{nf}}$ refers to Lemma 4.1.3 unless stated otherwise.

Example 4.1.4 (Normal form with clashing equalities)**.** $\langle[\mathtt{a}][\mathtt{c} \mapsto \mathtt{Z}]{\cdot}\mathtt{X} \approx_\alpha [\mathtt{b}]\mathtt{Y}\rangle_{\mathtt{nf}} =$

$$\{\{[\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b}){\cdot}\mathtt{Z}]\hat{}(\mathtt{a}\ \mathtt{b}){\cdot}\mathtt{X}\ {}_?\!\approx\!_?\ \mathtt{Y}, \mathtt{b}\#\mathtt{X}, \mathtt{c}\#\mathtt{X}\}, \{[\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b}){\cdot}\mathtt{Z}]\hat{}(\mathtt{a}\ \mathtt{b}){\cdot}\mathtt{X}\ {}_?\!\approx\!_?\ \mathtt{Y}, \mathtt{b}\#\mathtt{X}, \mathtt{b}\#\mathtt{Z}\}\}.$$

Intuitively, a *unifier* of the unification problem $[\mathtt{a}][\mathtt{c} \mapsto \mathtt{Z}]{\cdot}\mathtt{X}\ {}_?\!\approx\!_?\ [\mathtt{b}]\mathtt{Y}$ is any v-substitution σ such that $\Gamma \vdash [\mathtt{c} \mapsto (\mathtt{a}\ \mathtt{b}){\cdot}\mathtt{Z}\sigma]\hat{}(\mathtt{a}\ \mathtt{b}){\cdot}\mathtt{X}\sigma \approx_\alpha \mathtt{Y}\sigma$ for some freshness context Γ where $\Gamma \vdash \mathtt{b}\#\mathtt{X}\sigma, \mathtt{c}\#\mathtt{X}\sigma$ or $\Gamma \vdash \mathtt{b}\#\mathtt{X}\sigma, \mathtt{b}\#\mathtt{Z}\sigma$, possibly both. A formal definition follows.

Definition 4.1.5 (Unifier)**.** A v-substitution σ is called a **unifier** of a unification problem $Pr = \{s_i \,?\!\approx\!?\, t_i \mid i \in \mathrm{I}\}$ if and only if $\nabla \vdash s_i\sigma \approx_\alpha t_i\sigma$ for each $s_i \,?\!\approx\!?\, t_i \in Pr$ and some freshness context ∇. Pr is **unifiable** if there exists a unifier for it.

Primitive constraints included in a unification problem-in-context are not unifiable per se. There is an infinite number of v-substitutions making a predicate of form $a\#X$ true. The motivation of including primitive constraints in a unification problem-in-context is another, namely to provide soundness of unifiers for a set of unification constraints with respect to some freshness conditions Δ, that is, $\nabla \vdash s\sigma \approx_\alpha t\sigma$ and $\nabla \vdash \Delta\sigma$ for some consistent freshness context ∇, v-substitution σ and unification problem-in-context $\{s \,?\!\approx\!?\, t\} \cup \Delta$. Therefore, solving both unification problems and unification problems-in-context involves finding a pair of a v-substitution σ and a freshness context ∇ such that $\nabla \vdash Pr\sigma$.

Next, we define how v-substitutions and freshness contexts interact and employ normal form function $\langle\cdot\rangle_{\mathtt{nf}}$ from Lemma 4.1.3 to prove an useful property of such interaction.

Definition 4.1.6. The **action of v-substitutions σ on a freshness context** ∇, written $\nabla\sigma$, is defined as $\nabla\sigma \triangleq \{a\#\sigma(X) \mid X \in dom(\sigma), a\#X \in \nabla\}$

The definition above leads to the following straightforward property.

Property 4.1.7. *For any v-substitution σ and freshness context ∇, σ **satisfies** ∇* iff $\langle\nabla\sigma\rangle_{\mathtt{nf}} \neq \bot$.

Say a freshness context $\nabla\sigma$ is an **instance** of ∇ under σ for the case where $\langle\nabla\sigma\rangle_{\mathtt{nf}} \neq \bot$.

It is easy to see that, if σ satisfies ∇ and $\Delta \subseteq \nabla$ then σ satisfies Δ. Additionally, if $\langle\nabla\sigma\rangle_{\mathtt{nf}} \neq \bot$ and θ satisfies $\nabla\sigma$, then $(\sigma \circ \theta)$ satisfies ∇ and $(\nabla\sigma)\theta = \nabla(\sigma \circ \theta)$.

We are now ready to provide the definition of a solution to a unification problem.

4.1.3 Representation of unification solutions

Definition 4.1.8. A **solution** to a unification problem Pr is a pair $(\mathbf{F}, \sigma)$ consisting of a collection of freshness contexts $\mathbf{F}$ and a v-substitution σ satisfying, for each $\Delta \in \mathbf{F}$:

- $\Delta \vdash a\#r\sigma$ for each $a\#r \in Pr$,
- $\Delta \vdash s\sigma \approx_\alpha t\sigma$ for each $s \,?\!\approx\!?\, t \in Pr$,
- $\Delta \vdash \sigma \bullet \sigma \approx_\alpha \sigma$. We say σ is **idempotent**.

For the case where there is no $(\mathbf{F}, \sigma)$, the problem is said to be **unsolvable**.

In Definition 4.1.5 it was shown that there could be more than one freshness context ∇ for the same unifier σ. We have chosen to classify solutions by their unifiers, thus resulting in a pair of a unifier and a collection of freshness contexts from which the unification is derivable. A distinct structure for a solution could have been chosen where the pair would consist of a unifier and a corresponding freshness context. Then, solutions with distinct freshness contexts but sharing a common unifier would also be considered distinct. Our approach prioritises unifiers over freshness contexts since, as we explained before, freshness contexts provide soundness of unifiers. This becomes relevant in the following Chapter 5 where we aim to solve *pattern-matching problems*.

In the sequel, we omit the outer brackets if set $\mathbf{F}$ is a singleton. For instance, singletons $\{\varnothing\}, \{\nabla\}$ and $\{\bot\}$ are now represented as $\varnothing, \nabla$ and $\bot$ respectively.

A solution to a unification problem-in-context is a particular case of the definition above for unification problems as we show next.

Definition 4.1.9. A **solution** to a unification problem-in-context $(\nabla \vdash s) \mathrel{?}\approx_? (\Delta \vdash t)$ is a unifier σ such that there exists a collection of freshness contexts $\mathbf{F}$ such that $(\mathbf{F}, \sigma)$ is a solution to the unification problem $\{s \mathrel{?}\approx_? t\} \cup \nabla \cup \Delta$.

Properties and examples of unification solutions are defined considering only *idempotent solutions* to unification problems(-in-context), that is, solutions where the unifier σ is such that $\Delta \vdash \sigma \bullet \sigma \approx_\alpha \sigma$ for some freshness context Δ. This is a design choice since it is not strictly necessary.

Next, an example of a solution to a unification problem.

Example 4.1.10 (Unification solution)**.** There are many solutions to the unification problem

$$\{[\mathtt{c} \mapsto [\mathtt{a}]\mathtt{a}]\cdot \mathtt{X} \mathrel{?}\approx_? \mathtt{g}([\mathtt{b}]\mathtt{Y}, [\mathtt{a}]\mathtt{a})\}.$$

One solution is

$$(\{\mathtt{c\#Y}\}, [\mathtt{X} \mapsto \mathtt{g}([\mathtt{b}]\mathtt{Y}, [\mathtt{a}]\mathtt{a})]).$$

From now on we make no distinction between a unification problem and a unification problem-in-context unless stated otherwise. They are both referred to as unification problem Pr.

4.1.4 Properties of solutions to unification problems

Recall that a solution to a unification problem $\{s \mathrel{?}\approx_? t\} \cup \Gamma$ (Γ an arbitrary set of freshness constraints) is *any* v-substitution σ resolving $\Delta_i \vdash s\sigma \approx_\alpha t\sigma, \Gamma\sigma$ for one or more distinct freshness contexts Δ_i.

Example 4.1.11. Another solution to the unification problem given in Example 4.1.10 is

$$(\{\mathtt{c\#Y}\}, [\mathtt{X} \mapsto \mathtt{g}([\mathtt{b}]\mathtt{Y}, \mathtt{c})]).$$

Example 4.1.12. The unification problem

$$\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}$$

has a solution $\theta_1 = (\varnothing, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])$. Another solution is $\theta_2 = (\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])$. There is also solution $\theta_3 = (\{\mathtt{a\#Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])$ and so on.

It is left to the reader to check they are indeed solutions to the given unification problems. This is done using Definition 4.1.8. Next, a definition of set of all solutions to a unification problem.

Definition 4.1.13 (Set of all solutions to a unification problem)**.** Write $\mathscr{U}(Pr)$ for the **set of all unification solutions** to a given unification problem Pr.

We tend to drop the outer brackets of Pr in $\mathscr{U}(Pr)$ when enumerating constraints in Pr.

Accordingly, unification solutions in $\mathscr{U}(Pr)$ have a natural *partial* order defined by the relation $\leq$ on $\mathscr{U}(Pr)$. The relation $\leq$ is formally defined as follows.

Definition 4.1.14 (Instantiation ordering)**.** Define the **instantiation ordering** $\leq$ by $(\mathbf{F}, \sigma) \leq (\mathbf{F}', \sigma')$, if for each $\Delta \in \mathbf{F}'$ there exists both a $\nabla \in \mathbf{F}$ and a v-substitution θ such that:

- $\Delta \vdash \sigma \bullet \theta \approx_\alpha \sigma'$ and
- $\Delta \vdash \nabla\theta$.

Then, we say that solution $(\mathbf{F}, \sigma)$ is **more general** than solution $(\mathbf{F}', \sigma')$. We also say that $(\mathbf{F}', \sigma')$ is **an instance of** $(\mathbf{F}, \sigma)$.

Intuitively, Definition 4.1.14 specifies a partial order since $(\mathbf{F}, \sigma) \leq (\mathbf{F}', \sigma')$ means that $(\nabla, \sigma) \leq_\theta (\Delta, \sigma')$ for each $\Delta \in \mathbf{F}'$ and some $\nabla \in \mathbf{F}$ along with some v-substitution θ, as given in Definition 1.2.24 for standard terms.

There are solutions to a problem where neither of them is an instance of the other. We define this notion below.

Definition 4.1.15 (Incomparable solutions)**.** Call a pair of solutions $(\mathbf{F}, \sigma), (\mathbf{F}', \sigma')$ **incomparable** when there is no θ such that $(\mathbf{F}, \sigma) \leq (\mathbf{F}', \sigma')$ or $(\mathbf{F}', \sigma') < (\mathbf{F}, \sigma)$.

Example 4.1.16. From the solutions in Example 4.1.12 we observe that,
$(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}]) \leq (\varnothing, [\mathtt{X} \mapsto \mathtt{b}; \mathtt{Z} \mapsto \mathtt{a}])$ since $(\varnothing, [\mathtt{X} \mapsto \mathtt{b}; \mathtt{Z} \mapsto \mathtt{a}])$ is an instance of $(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])$ by means of $\theta = [\mathtt{Z} \mapsto \mathtt{a}]$.

The additional pair of solutions $(\varnothing, [\mathtt{X} \mapsto \mathtt{c}; \mathtt{Z} \mapsto \mathtt{c}])$ and $(\varnothing, [\mathtt{X} \mapsto \mathtt{U}; \mathtt{Y} \mapsto \mathtt{b}; \mathtt{Z} \mapsto \mathtt{U}])$ ($\mathtt{U}$ a new variable) is incomparable.

The following technical lemma states that the instantiation ordering is preserved under v-substitution application. Afterwards, Lemma 4.1.18 states that the instantiation ordering is invariant under α-equivalence. Both results follow by Definition 4.1.14.

Lemma 4.1.17. *If* $(\mathbf{F}, \sigma) \leq (\mathbf{F}', \sigma')$ *then* $(\mathbf{F}, \theta \bullet \sigma) \leq (\mathbf{F}', \theta \bullet \sigma')$.

Proof. The result follows by Lemma 2.5.13 and Definition 4.1.14. The full proof can be found in Appendix B. ■

Lemma 4.1.18. *Suppose* $\nabla \vdash \sigma \approx_\alpha \sigma'$ *for all* $\nabla \in \mathbf{F}'$. *If* $(\mathbf{F}, \theta) \leq (\mathbf{F}', \sigma)$ *then* $(\mathbf{F}, \theta) \leq (\mathbf{F}', \sigma')$.

Proof. The result follows by Definition 4.1.14. The full proof can be found in Appendix B. ■

In (Urban et al., 2004), it was shown that the property of *unicity* of solutions holds for standard nominal terms, that is, if solutions exist for a unification problem, there is one solution that it is more general than any other solution up to variable renaming. Then, any other solution is an instance of the most general one, built by v-substitution composition and satisfiability of the freshness context. This is not the case for nominal terms in the presence of a-substitutions where there may be *infinitely* many principal solutions unifying a pair of terms. In fact, nominal unification with a-substitution is undecidable for the general case. This important negative result is discussed in depth in Section 4.3. Prior, we begin by providing a notion of *most general* solution to a unification problem; this is done in the next section.

4.2 Complete Set of Solutions of a Unification Problem

This section is concerned with the formal specification of principal solutions to unification problems in the extended nominal framework. It begins by providing a definition of principal solution to a unification problem. It follows by showing the issues that arise when searching for most general solutions in the presence of a-substitutions. The section ends with an example of a unification problem having infinitary principal solutions.

4.2.1 Principal solutions

We have shown how some unification problems may have more than one solution. In fact, there may be an infinite number of solutions unifying a pair of terms. Take for instance Example 4.1.12, every atom $n \in (\mathscr{A} \setminus \{a\})$ provides a solution $(\varnothing, [\mathtt{X} \mapsto \mathtt{n}; \mathtt{Z} \mapsto \mathtt{n}])$ to the unification problem. We have already established how a partial order can be defined among solutions. It is of interest to determine, when possible, which of the solutions to a problem are *principal* ones. Then, if the problem enjoys uniqueness of most general solutions, a principal solution is a least element in the ordering so that any other solution becomes an instance of the principal one, as described in Definition 4.1.14. Otherwise, there may be more than one principal solution in the ordering such that both solutions do not share a precedence relationship, that is, they are incomparable as stated in Definition 4.1.15. Take for instance solution θ_3 from Example 4.1.12, applying Definition 4.1.14 we observe that for each $n \in (\mathscr{A} \setminus \{a\})$, $(\varnothing, [\mathtt{X} \mapsto \mathtt{n}; \mathtt{Z} \mapsto \mathtt{n}])$ is an instance of θ_3 but θ_1 is not, neither is θ_3 an instance of θ_1. They are both incomparable. Hence, there may be more than one principal solution to the unification problem given in Example 4.1.12.

In standard nominal terms, there exists a unique principal solution in the set of solutions to any unification problem. This is not the case for nominal terms with a-substitutions, as we show in the following example.

Example 4.2.1. The pair of v-substitutions $[X \mapsto c]$ and $[X \mapsto a]$ are the only unifiers to the unification problem

$$\{[a \mapsto c] \cdot X \ ?\approx_? \ c\}$$

such that $(\varnothing, [X \mapsto c])$ and $(\varnothing, [X \mapsto a])$ are solutions to the problem.

There is no v-substitution θ such that $(\varnothing, [X \mapsto c]) \leq (\varnothing, [X \mapsto a])$ or $(\varnothing, [X \mapsto a]) < (\varnothing, [X \mapsto c])$, that is, using Definition 4.1.15 we observe that both solutions are in fact incomparable.

We need a definition of principality of solutions before continuing commenting on the particularities of the example above.

Definition 4.2.2. A **principal**, or **most general, solution** to a problem Pr, $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr)$, is one such that, given any other solution $(\mathbf{F}', \theta)$ from the partially ordered set $\mathscr{U}(Pr)$, it is not the case that $(\mathbf{F}', \theta) < (\mathbf{F}, \sigma)$ for any v-substitution τ. Then, we call σ a **most general unifier** to Pr

In Example 4.2.1, both solutions $(\varnothing, [X \mapsto c])$ and $(\varnothing, [X \mapsto a])$ are principal ones to the unification problem $\{[a \mapsto c] \cdot X \ ?\approx_? \ c\}$ since, for any other solution to the problem, $(\mathbf{F}, \theta)$,

there is no v-substitution τ such that $(\mathbf{F},\theta) < (\varnothing,[X \mapsto c])$ or $(\mathbf{F},\theta) < (\varnothing,[X \mapsto a])$. This is the case as a result of both unifiers being unique to the problem (modulo renaming variables) and both collection of sets over freshness contexts being singletons containing an empty freshness context. Accordingly, the role of principality must be taken on, in general, by a set of principal solutions.

Below, we provide a formal definition of the complete set of principal solutions to a unification problem.

Definition 4.2.3 (Complete set of principal solutions)**.** Given a unification problem Pr, we say that $W = \{(\mathbf{F}_i,\theta_i) \mid i \in \mathrm{I}\}$ is a **complete set of principal solutions** to Pr *iff*

1. $\forall(\mathbf{F}_i,\theta_i) \in W$, $\mathscr{D}om(\theta_i) \subseteq V(Pr)$,
2. $W \subseteq \mathscr{U}(Pr)$,
3. $\forall(\mathbf{F},\sigma) \in \mathscr{U}(Pr), \exists(\mathbf{F}_i,\theta_i) \in W : (\mathbf{F}_i,\theta_i) \leq (\mathbf{F},\sigma)$.

By satisfying each of the three statements from Definition 4.2.3 we observe that, the complete set of principal solutions to the unification problem from Example 4.2.1 contains only $(\varnothing,[X \mapsto c])$ and $(\varnothing,[X \mapsto a])$ as elements of the solution set.

4.2.2 Infinitary unification

Similarly to second-order unification (e.g. (Levy and Veanes, 2000)), there is an infinite number of principal solutions to some nominal unification problems containing variables with suspended a-substitution. Take for instance terms $s = [c \mapsto f(a,b)]\cdot X$ and $t = f(a,[c \mapsto b]\cdot X)$ and the unification problem

$$\{[c \mapsto f(a,b)]\cdot X \stackrel{?}{\approx}_? f(a,[c \mapsto b]\cdot X)\}. \tag{4.1}$$

V-substitution $\sigma_0 = [X \mapsto c]$ is a unifier of (4.1) since $\vdash s\sigma_0 \approx_\alpha t\sigma_0 \approx_\alpha f(a,b)$; v-substitution $\sigma_1 = [X \mapsto f(a,c)]$ is another unifier of (4.1) since $\vdash s\sigma_1 \approx_\alpha t\sigma_1 \approx_\alpha f(a,f(a,b))$. Another unifier is $\sigma_2 = [X \mapsto f(a,f(a,c))]$ since $\vdash s\sigma_2 \approx_\alpha t\sigma_2 \approx_\alpha f(a,f(a,b))$. In fact, there is an infinite number of solutions of the form

$$\sigma_n = [X \mapsto f(a,f(a,\ldots,f(a,c)\cdots))]$$

where n is the number of occurrences of function symbol f and atom a in term $\sigma_n(X)$. It is easy to see that each of the generated solutions is a principal one when considering only idempotent solutions, since there is only variable X in the unification problem and each

unifier contains a ground term. Thus, the unification problem (4.1) has an infinite number of principal solutions.

Next, a proof of undecidability for unification of extended terms. We do not aim to find a complete semi-decision procedure for nominal unification with a-substitution as it is done for higher-order unification as in (Pietrzykowski, 1973) or (Dowek et al., 2000), for instance. Our goal is, instead, to restrict the class of unification problems in order to recover the property of unique principal solution to a unification problem so that a unification algorithm can be used as a matching tool for the rewriting of extended terms in Chapter 6. As we demonstrate in Chapter 5, such goal is achieved by both adjusting the specification of the unification theory and finding a suitable class of unification problems for which unification of extended terms is decidable.

4.3 Undecidability of Extended Nominal Unification

To prove undecidability of extended nominal unification, we follow closely Goldfarb's *undecidability proof for second-order unification problems* (Goldfarb, 1981). In his seminal paper, Goldfarb uses *Hilbert's tenth problem*, proved undecidable in (Matiyasevich, 1970), to prove also that second-order unification is undecidable. Goldfarb does not use variable binding operators like λ to generate *Diophantine equations* but sophisticated number-theoretic constructs to build a unification problem for which any *ground unifier* simulates multiplication; similarly for the simulation of addition. Thus, our objective is also to reduce Hilbert's tenth problem to extended nominal unification.

In Goldfarb's proof, the signature of the term language contains a binary function symbol g and a pair of constant symbols a, b where the former is used to represent sequences of the latter. Our proof of undecidability replaces the pair of constant symbols by a pair of unabstracted atoms (recall that an atom term is also a ground term) and the generation of sequences of unabstracted atoms is obtained by the use of tuples (see Definition 2.1.3). For instance, natural number n is represented as lambda-term $\lambda x.g(a,g(a,\dots g(a,x)))$ and denoted by $\overline{n}$ with n occurrences of g and a. Such terms are called **Goldfarb numbers** and they are a revision on Church's number representation by λ-terms, $\lambda x.\lambda f.f(f(\dots f(x)))$, where the abstraction on f is dropped so that the representation is of second-order type. Goldfarb numbers are denoted as nominal terms $(a,(a,\dots(a,c)))$ with n occurrences of $()$ and a to represent natural number n, where $a, c \in \mathscr{A}$. Goldfarb numbers are exactly those that solve the extended nominal unification problem

$$\{(a,[c \mapsto a]\cdot F) \mathrel{?}\approx\mathrel{?} [c \mapsto (a,a)]\cdot F\} \tag{4.2}$$

as adapted from (Goldfarb, 1981).

We begin by recalling the definition of Diophantine equations.

4.3.1 Diophantine equations

Definition 4.3.1. A Diophantine equation is an equation of the form

$$P(X_1,\ldots,X_n) = Q(X_1,\ldots,X_n)$$

where $P(X_1,\ldots,X_n), Q(X_1,\ldots,X_n)$ are a pair of polynomials with natural coefficients. Then, a solution to a Diophantine equation is a tuple of natural numbers $m_1,\ldots,m_n$ such that

$$P(m_1,\ldots,m_n) = Q(m_1,\ldots,m_n).$$

Example 4.3.2. The following equation is Diophantine:

$$x^2 + y^2 = z^2$$

and it is called a *Pythagorean triple* (Silverman, 1997). There are infinitely many solutions (x,y,z) to such Diophantine equation. A well-known *primitive* solution to such equation is $(3,4,5)$, another solution is $(3\times k, 4\times k, 5\times k)$ for any $k > 1$.

In today's terminology, Hilbert's tenth problem is a decision problem demanding a single universal method (thus a Turing machine) that could be applied to any Diophantine equation with any number of unknowns and integer coefficients to decide, in a finite number of steps, whether or not a finite set of integer solutions to the polynomial equation exists.

The decision problem was resolved in the negative by Yuri Matiyasevitch in 1970 (Matiyasevich, 1970). Following Goldfarb's steps in (Goldfarb, 1981) for higher-order unification, we observe that a reduction of Hilbert's tenth problem requires to represent natural numbers, addition and multiplication in terms of extended nominal unification. Such reduction of the decision problem also leads to a negative result on unification of extended nominal terms. Using the grammar of extended nominal terms in Definition 2.1.3 we define a pair of unification problems $Pr^+, Pr^\times$ for which *any ground unifier* to Pr^+ *simulates* addition and *any ground unifier* to $Pr^\times$ *simulates* multiplication. Then, applying both unification problems, one is able to define an effective method to generate Diophantine equations and thus providing an undecidability proof by transfering the result from (Matiyasevich, 1970). This is done in Theorem 4.3.8.

We begin by providing a definition of the extended nominal term language.

4.3.2 Term Language L

Definition 4.3.3 (Term language L)**.** The term language L is a triple $(\Sigma_L, \mathscr{X}, \mathscr{A})$ of disjoint sets where Σ_L is an empty signature and $\mathscr{X}, \mathscr{A}$ are the countable sets of variables and atoms respectively as described in Section 2.1.1. Then, the set of all terms in language L is defined as the set of all extended nominal terms generated over a grammar equivalent to that given in Definition 2.1.3 except abstraction terms are not part of the grammar. We refer to such terms as **L-terms**.

Informally, language L contains a restricted class of extended nominal terms with no function symbols and abstraction terms. This constraint on the grammar of extended terms and the restriction to ground unifiers allows us to state that any unifier to Lemma 4.3.6 and Lemma 4.3.7 simulates addition and multiplication respectively. Intuitively, if unification of L-terms is proven undecidable, it is easy to see that undecidability also holds for the general case.

In order to improve readability, nested tuples are sometimes flattened. This is shown below.

Remark 4.3.4. In the sequel, for all L-terms $t_1, \ldots, t_{n+1}$ with $n > 0$, $(t_1, \ldots, t_{n+1}) = (t_1, (t_2, \ldots, t_{n+1}))$. Note also that, $(t_1, \ldots, t_n, t_{n+1}, t_{n+2}) = (t_1, \ldots, t_n, (t_{n+1}, t_{n+2}))$ and, similarly, $(t_1, \ldots, t_n, t_{n+1}) = ((t_1, \ldots, t_n), t_{n+1})$. Then, given an atom $a \in \mathscr{A}$, L-term t and integer n such that $n \geq 0$, let $[n, a, t]$ be the L-term defined as follows: $[0, a, t]$ is t and similarly, $[n+1, a, t]$ is $(a, [n, a, t])$.

Intuitively, it follows that $[n, a, t] = [m, a, t]$ if and only if $m = n$.

The following property is easily derived from Remark 4.3.4.

Property 4.3.5. *Given an atom $a \in \mathscr{A}$,* L*-term t and integers n, m such that $n, m \geq 0$,*

- $[n, a, [m, a, t]] = [n + m, a, t]$.

Proof. It follows directly from Remark 4.3.4 and the structure of L-terms. ■

Following a notation closer to that of Goldfarb's term language, in the sequel, L-terms of form $[n, a, t]$ are denoted as $\overline{n}_a t$ for any atom $a \in \mathscr{A}$, L-term t and integer n such that $n \geq 0$. Then, $[n, a, [m, a, t]]$ is thus denoted as $\overline{n + m}_a t$.

4.3.3 Constructing a unification problem to simulate addition

One can easily derive a pair of L-terms from the second-order unification problem given in (Goldfarb, 1981) so that any ground unifier to such pair simulates addition. Observe that addition in (Goldfarb, 1981) was represented following Church's notation for λ-term

$add = \lambda n.\lambda m.\lambda x.n(m(x))$. We adapt the notation for our extended framework; this is done in the lemma below.

Lemma 4.3.6 (Simulating addition)**.** *Let* $Pr^{+} = \{[c_1 \mapsto [c_1 \mapsto a]\cdot F_2]\cdot F_1 \stackrel{?}{\approx}_{?} [c_1 \mapsto a]\cdot F_3\}$. *For all* $m,n,p \geq 0$*, there exists a ground unifier* θ *for* Pr^{+} *such that* $\{[F_1 \mapsto \overline{n}_a c_1; F_2 \mapsto \overline{m}_a c_1; F_3 \mapsto \overline{p}_a c_1]\} \subseteq \theta$ *if and only if* $p = m+n$.

Proof. Let $m,n,p \geq 0$.

1. *Only if.* Let θ be a ground unifier of Pr^{+} containing v-substitution $[F_1 \mapsto \overline{n}_a c_1; F_2 \mapsto \overline{m}_a c_1; F_3 \mapsto \overline{p}_a c_1]$. Then, $\overline{n}_a(\overline{m}_a a) \approx_\alpha \overline{p}_a a$. By Property 4.3.5, $\overline{m}_a(\overline{n}_a a) = \overline{m+n}_a a$, and by the transitive property from Theorem 2.5.8, we have $\overline{m+n}_a a \approx_\alpha \overline{p}_a a$. The result then follows.

2. *If.* Let $p = m+n$ and let θ be a unifier of Pr^{+} containing v-substitution $[F_1 \mapsto \overline{n}_a c_1; F_2 \mapsto \overline{m}_a c_1; F_3 \mapsto \overline{p}_a c_1]$. The result follows similarly to the previous case and thus omitted.

■

4.3.4 Constructing a unification problem to simulate multiplication

Next, a unification problem to simulate multiplication is defined. Once again, the lemma and its proof are closely based on their counterpart in (Goldfarb, 1981).

Lemma 4.3.7. *Let* $Pr^{\times} = \{s_1 \stackrel{?}{\approx}_{?} s_2\}$ *where*
$s_1 = [c_1 \mapsto a; c_2 \mapsto b; c_3 \mapsto (([c_1 \mapsto a]\cdot F_3, [c_1 \mapsto b]\cdot F_2), a)]\cdot G$ *and*
$s_2 = ((a,b), [c_1 \mapsto [c_1 \mapsto a]\cdot F_1; c_2 \mapsto \overline{1}_b b; c_3 \mapsto a]\cdot G)$.

For all $m,n,p \geq 0$*, there is a ground unifier* θ *for* $Pr^{\times}$ *such that* $\sigma = [F_1 \mapsto \overline{m}_a c_1; F_2 \mapsto \overline{n}_b c_1; F_3 \mapsto \overline{p}_a c_1]$ *and* $\sigma \subset \theta$ *if and only if* $p = m \times n$.

Proof. Let $m,n,p \geq 0$ and define a pair of a-substitutions thus:

$$\phi_{s_1} = [c_1 \mapsto a; c_2 \mapsto b; c_3 \mapsto ((\overline{p}_a a, \overline{n}_b b), a)] \qquad \phi_{s_2} = [c_1 \mapsto \overline{m}_a a; c_2 \mapsto \overline{1}_b b; c_3 \mapsto a].$$

Now, notice that, for any L-term u and ground v-substitution θ containing both σ and $[G \mapsto u]$ we have

$$s_1\theta \approx_\alpha u\phi_{s_1} \quad \text{and also} \quad s_2\theta \approx_\alpha ((a,b), u\phi_{s_2}). \tag{4.3}$$

Following this representation, we provide a general pattern for the structure of u to be used in the rest of the proof. Let $t_k = (\overline{m \times k}_a c_1, \overline{k}_b c_2)$ for each $k \geq 0$. Then, observe that, by

application of the Transitive property from Theorem 2.5.8, it is the case that

$$t_{k+1}\phi_{s_1} \approx_\alpha (\overline{m \times (k+1)}_a c_1, \overline{k+1}_b c_2) \approx_\alpha t_k \phi_{s_2}. \tag{4.4}$$

- *If.* Let $p = m \times n$ and $\theta = (\sigma \bullet [G \mapsto u])$. We want to prove that θ is a unifier of $\{s_1 \ ?\!\approx\!?\ s_2\}$. Then, there are two cases to examine, the case where $n = 0$ and the case where $n > 0$.

  1. (Case $n = 0$). If $n = 0$ then $u = c_3$ so that, following (4.3), the Transitive property from Theorem 2.5.8 and Definition 2.3.2 we have, $s_1\theta \approx_\alpha u\phi_{s_1} \approx_\alpha ((\overline{p}_a a, \overline{n}_b b), a) \approx_\alpha ((a,b),a) \approx_\alpha ((a,b), u\phi_{s_2}) \approx_\alpha s_2\theta$.
  2. (Case $n > 0$). If $n > 0$ then $u = (t_0, \ldots, t_{n-1}, c_3)$ so that, following (4.3), the Transitive property from Theorem 2.5.8 and Definition 2.3.2 we have, $s_1\theta \approx_\alpha u\phi_{s_1} \approx_\alpha (t_0\phi_{s_1}, \ldots, t_{n-1}\phi_{s_1}, \phi_{s_1}(c_3)) \approx_\alpha (t_0\phi_{s_1}, \ldots, t_{n-1}\phi_{s_1}, ((\overline{p}_a a, \overline{n}_b b), a))$. Since $p = m \times n$, it is the case that $(\overline{p}_a a, \overline{n}_b b) \approx_\alpha t_n\phi_{s_1}$ such that, by application of Transitive property from Theorem 2.5.8, $(t_0\phi_{s_1}, \ldots, t_{n-1}\phi_{s_1}, ((\overline{p}_a a, \overline{n}_b b), a)) \approx_\alpha (t_0\phi_{s_1}, \ldots, t_{n-1}\phi_{s_1}, t_n\phi_{s_1}, a)$. Also, since $t_0\phi_{s_1} \approx_\alpha (\overline{m \times 0}_a a, \overline{0}_b b)$ by application of Definition 2.3.2, it is the case that, using the Transitive property from Theorem 2.5.8 we obtain, $(t_0\phi_{s_1}, \ldots, t_{n-1}\phi_{s_1}, t_n\phi_{s_1}, a) \approx_\alpha (a,b), t_1\phi_{s_1}, \ldots, t_{n-1}\phi_{s_1}, t_n\phi_{s_1}, a) \approx_\alpha ((a,b), (t_1\phi_{s_1}, \ldots, t_n\phi_{s_1}, a))$.

     Now, following (4.3), the Transitive property from Theorem 2.5.8 and Definition 2.3.2 we have, $s_2\theta \approx_\alpha ((a,b), u\phi_{s_2}) \approx_\alpha ((a,b), (t_0\phi_{s_2}, \ldots, t_{n-1}\phi_{s_2}, a))$. Further, following (4.4) we observe that, $(t_0\phi_{s_2}, \ldots, t_{n-1}\phi_{s_2}, a) \approx_\alpha (t_1\phi_{s_1}, \ldots, t_n\phi_{s_1}, a)$ such that, by application of the Transitive property from Theorem 2.5.8 we are able to derive $s_2\theta \approx_\alpha ((a,b), u\phi_{s_2}) \approx_\alpha ((a,b), (t_1\phi_{s_1}, \ldots, t_n\phi_{s_1}, a)) \approx_\alpha u\phi_{s_1} \approx_\alpha s_1\theta$ and we are done.

  Hence, θ is indeed a unifier of $\{s_1 \ ?\!\approx\!?\ s_2\}$.

- *Only if.* Suppose θ is a unifier of $Pr^\times$ such that $\sigma \subset \theta$. Then, θ must also contain v-substitution $[G \mapsto u]$ for some L-term u and we have,

$$s_1\theta \approx_\alpha u\phi_{s_1} \approx_\alpha ((a,b), u\phi_{s_2}) \approx_\alpha s_2\theta. \tag{4.5}$$

  Therefore, observe that either $u = c_3$ or $u = (v, v')$ for L-terms v, v'.

  1. Suppose $u = c_3$. Then, by (4.5) and Definition 2.3.2 we have, $s_1\theta \approx_\alpha ((\overline{p}_a a, \overline{n}_b b), a) \approx_\alpha ((a,b),a) \approx_\alpha s_2\theta$. Hence, it can only be that $p = n = 0$ and thus $p = m \times n$ as expected.

2. Suppose $u = (v, v')$ for some L-terms v, v'. Further, let k be the largest integer such that $u = (v_0, \ldots, v_k)$ for $k > 0$ and some L-terms $v_0, \ldots, v_k$. Then, by (4.5) and Definition 2.3.2 we have, $s_1\theta \approx_\alpha (v_0\phi_{s_1}, \ldots, v_k\phi_{s_1}) \approx_\alpha ((a,b), v_0\phi_{s_2}, \ldots, v_k\phi_{s_2}) \approx_\alpha s_2\theta$. Then, using Definition 2.2.5 we observe the following:

   (a) $v_0\phi_{s_1} \approx_\alpha (a,b)$,

   (b) $v_i\phi_{s_1} \approx_\alpha v_{i-1}\phi_{s_2}$ for $0 < i < k$ and

   (c) $v_k\phi_{s_1} \approx_\alpha (v_{k-1}\phi_{s_2}, v_k\phi_{s_2})$.

   By (a) we have $v_0 = (c_1, c_2)$, that is, $v_0 = t_0$ (recall $t_k = (\overline{m \times k}_a c_1, \overline{k}_b c_2)$ $(k \geq 0)$ as stated at the beginning of the proof). And then by (b) and application of the Transitive property from Theorem 2.5.8 it follows that, $v_0\phi_{s_2} \approx_\alpha t_0\phi_{s_2} \approx_\alpha (\overline{m \times 1}_a a, \overline{1}_b b) \approx_\alpha t_1\phi_{s_1} \approx_\alpha v_1\phi_{s_1}$. Hence it must be that $v_1 \approx_\alpha t_1$. Applying the same reasoning we can repeatedly infer the rest of (b), that is, for cases $1 < i < k$ such that $v_2 \approx_\alpha t_2, \ldots, v_{k-1} \approx_\alpha t_{k-1}$. Now, since $v_{k-1} \approx_\alpha t_{k-1}$, by the Transitive property from Theorem 2.5.8 applied to (c) we have, $v_k\phi_{s_1} \approx_\alpha (t_{k-1}\phi_{s_2}, v_k\phi_{s_2})$ and by Equation 4.4 and the Transitive property from Theorem 2.5.8, $(t_{k-1}\phi_{s_2}, v_k\phi_{s_2}) \approx_\alpha (t_k\phi_{s_1}, v_k\phi_{s_2}) \approx_\alpha ((\overline{m \times k}_a a, \overline{k}_b b), v_k\phi_{s_2})$. Then, either $v_k = c_3$ or $v_k = (v, v')$ for some L-terms v, v'. However, if $v_k = (v, v')$ we have $u \approx_\alpha (v_0, \ldots, v_{k-1}, (v, v')) \approx_\alpha (v_0, \ldots, s_{k-1}, v, v')$, contrary to the choice of integer k being the largest. Hence, it is the case that $v_k = c_3$ and then, using the Transitive property from Theorem 2.5.8 we obtain, $v_k\phi_{s_1} \approx_\alpha ((\overline{p}_a a, \overline{n}_b b), a) \approx_\alpha ((\overline{m \times k}_a a, \overline{k}_b b), v_k\phi_{s_2})$. Thus, it is the case that $k = n$ and we are able to conclude by stating that $p = m \times n$.

■

Finally, we are able to provide a proof of undecidability for extended nominal unification by describing a method to reduce *Hilbert's tenth problem* to a nominal unification problem using the term language from Definition 4.3.3 and Lemmas 4.3.6 & 4.3.7. This is done next.

4.3.5 Reducing Hilbert's tenth problem to unification of extended terms

Finally, we prove the undecidability of extended nominal unification using the term language from Definition 4.3.3, Lemma 4.3.6 & Lemma 4.3.7, as follows.

Note that every Diophantine equation as in definition 4.3.1 can be decomposed into a system of equations of the form: (below, m denotes a natural number)

$$X_i + X_j = X_k, \qquad X_i \times X_j = X_k, \qquad X_i = m.$$

Now, we associate a unification problem with each such system, containing

- for each X_i, a unification problem as given in Equation 4.2;
- for each $X_i = m$, the equation $X_i = \overline{m}_a$;
- for each $X_i + X_j = X_k$, the unification problem used to define addition in Lemma 4.3.6;
- for each $X_i \times X_j = X_k$, the unification problem used to define multiplication in Lemma 4.3.7.

We have thus an encoding of Hilbert's tenth problem.

Theorem 4.3.8. *There is an effective method that reduces Hilbert's tenth problem to the nominal unification problem for* L.

Proof. Let H be any finite set of equations having forms $X_i + X_j = X_k$, $X_i \times X_j = X_k$, and $X_i = C_j$ where X_i, X_j, X_k are numerical variables and C_j are numerical constants for $i, j, k \in \mathtt{I}$. A solution for H is an assignment of natural numbers to the numerical variables which makes every equation in H true. By using the methods to simulate addition and multiplication from Lemma 4.3.6 and Lemma 4.3.7 respectively, it suffices to construct a unification problem Pr^H for pairs of L-terms such that *there is a ground unifier for Pr^H if and only if H has a solution*.

Suppose $X_1, \ldots, X_q$ ($q \geq 1$) are all the numerical variables in H. Then, L-terms in Pr^H will contain nominal variables $F_1, \ldots, F_q, G_i$ ($i \in \mathtt{I}$). Now, let Pr^H contain the following unification pairs:

1. for each i, $1 \leq i \leq q$, $(\overline{1}_a[c_1 \mapsto a] \cdot F_i \mathrel{?\approx?} [c_1 \mapsto \overline{1}_a a] \cdot F_i) \in Pr^H$;
2. for all i and j such that $(X_i = C_j) \in H$, $([c_1 \mapsto a] \cdot F_i \mathrel{?\approx?} \overline{v_j}_a) \in Pr^H$ where v_j is the numerical value of C_j;
3. for all i, j, k such that $X_i + X_j = X_k$ is a member of H, $([c_1 \mapsto [c_1 \mapsto a] \cdot F_j] \cdot F_i \mathrel{?\approx?} [c_1 \mapsto a] \cdot F_k) \in Pr^H$ (observe that this pair is equal to Pr^+ from Lemma 4.3.6 with variables F_1, F_2, F_3 renamed to F_i, F_j, F_k respectively);
4. for all i, j, k such that $X_i \times X_j = X_k$ is a member of H, we have a unification pair obtained from the pair $s_1 \mathrel{?\approx?} s_2$ as given in Lemma 4.3.7 by renaming the subscripts on its variables as follows: F_1 is renamed F_i, F_2 is renamed F_j and F_3 is renamed F_k and G is renamed $G_{2^i 3^j 5^k}$.

   Now we a ready to prove the method above. Let θ be a ground unifier for Pr^H, that is, $Pr^H\theta$. By (1) and Definition 4.3.1, for each X_i there is a natural number n_i such that $[F_i \mapsto \overline{n_i}_a c_1] \in \theta$, $1 \leq i \leq q$. Our claim is that $n_1, \ldots, n_q$ is also a solution for H. This is shown as follows: by (2), if $X_i = C_j$ is in H, then $\overline{n_i}_a \approx_\alpha ([c_1 \mapsto a] \cdot F_i)\theta \mathrel{?\approx?} \overline{v_j}_a$ so that

$n_i = v_j$. By (3), if $X_i + X_j = X_k$ is in H then, $([c_1 \mapsto [c_1 \mapsto a]\cdot F_j]\cdot F_i)\theta \approx_\alpha ([c_1 \mapsto a]\cdot F_k)\theta$ and, since $[F_i \mapsto \overline{n_i}_a c_1; F_j \mapsto \overline{n_j}_a c_1; F_k \mapsto \overline{n_k}_a c_1] \subseteq \theta$, using Lemma 4.3.6 we have $n_i + n_j = n_k$. And similarly by (4), if $X_i \times X_j = X_k$ is in H then, since $[F_i \mapsto \overline{n_i}_a c_1; F_j \mapsto \overline{n_j}_a c_1; F_k \mapsto \overline{n_k}_a c_1] \subseteq \theta$, using Lemma 4.3.7 we have $n_i \times n_j = n_k$ and we are done.

Conversely, suppose the assignment of $n_1, \ldots, n_q$ to $X_1, \ldots, X_q$ is a solution for H. Let θ contain $[F_i \mapsto \overline{n_i}_a c_1]$ for each i, $1 \leq i \leq q$. Then, θ is a unifier for each pair of L-terms specified in (1)-(3) (using Lemma 4.3.6 in (3)). Now assume $X_i \times X_j = X_k$ is in H so that $n_i \times n_j = n_k$. By Lemma 4.3.7 there exists a L-term u such that if θ contains $[G_l \mapsto u]$ ($l = 2^i 3^j 5^k$), then θ is a unifier for the unification pair specified in 4.3.7. ■

Corollary 4.3.9. *Unification of nominal terms extended with atom substitution is undecidable.*

Proof. It follows by the undecidability proof of Hilbert's tenth problem (Matiyasevich, 1970), Theorem 4.3.8 and the fact that term language L is generated over the grammar of extended nominal terms (see Definition 2.1.3). ■

4.3.6 On non-uniqueness of most general solutions

As previously discussed, it is not the objective of this thesis to design a procedure to provide a complete set of principal solutions to any unifiable problem. Nevertheless, it is of interest to determine whether the unification theory can be adjusted in order to recover the property of uniqueness of most general solutions. However, the restrictions required to restore such property involve reducing the extended unification theory to its non-extended counterpart, as a result of a-substitutions being the main source of non-determinism of results. A clear example is given in Example 4.2.1, where it was shown that the pair of unifiers are principal solutions. An attempt to restrict the set of unifiable terms, considering only unification constraints with at least one occurrence of each variable symbol having a fixed position and trivial a-substitutions also proves unsuccessful. Take for instance the unification constraint $([\mathtt{a} \mapsto \mathtt{c}]\cdot \mathtt{X}, \mathtt{X}) \;{}_?{\approx}_?\; (\mathtt{c}, \mathtt{Y})$. Both variable symbols $\mathtt{X}, \mathtt{Y}$ have fixed occurrences with trivial moderation. However, the pair of solutions $(\varnothing, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto \mathtt{a}])$ and $(\varnothing, [\mathtt{X} \mapsto \mathtt{c}; \mathtt{Y} \mapsto \mathtt{c}])$ to such unification constraint are also principal; this provides a counterexample to our initial assumption on the validity of the constraint restriction. Nonetheless, there is a form of unification where such constraint restriction leads to the generation of unique principal solutions to unification problems; it involves considering terms on one side of symbol ${}_?{\approx}_?$ as constants. Then, instantiation is triggered solely on variable symbols encountered on the

LHS of $?\approx?$. This procedure, commonly known as *matching*, is instrumental on mechanising the rewriting of terms with respect to a set of rewrite rules. Both, rewriting and a decidable unitary unification algorithm are themes that we introduce in the following chapters and therefore we postpone the discussion until then.

4.4 Conclusion

In this chapter we have dealt with the problem of unifying nominal terms extended with a-substitutions. In Section 4.1, a definition of the unification problem for extended nominal terms and the structure of its solution was given, along with some important properties about such solutions which will be of use in the next chapter. Then, the notion of most general solution to a unification problem was defined in Section 4.2. There, it was shown that suspended a-substitutions on variables add complexity to the finding of solutions to unification problems, thus leading to the generation of one or more principal solutions to any unifiable problem. In fact, for some particular unification problems there may be an infinite number of principal solutions. Moreover, similarly to second-order unification, there are cases where unification of extended terms is undecidable. This is the main result of this chapter and it is discussed in Section 4.3. This negative result on unification of extended terms is proven by adapting the seminal work on undecidability of second-order unification by Goldfarb (Goldfarb, 1981) to nominal theory, that is, providing an effective method, in Theorem 4.3.8, to reduce Hilbert's tenth problem to unification of extended terms. Finally, we presented an overview on the issues to be overcome when defining a unification algorithm for the matching of extended terms. That is, we pondered upon the minimum restrictions to be imposed both on the unification theory and on the set of extended terms in order to make unification both decidable and unitary. Such algorithm will be applied later, in Chapter 6, when defining rewriting of extended terms. Finding a unification algorithm to match extended terms is the topic of the next chapter.

Chapter 5

A Unitary Unification Algorithm For Simple Matching Problems

In the previous chapter we have shown nominal unification of extended terms to be infinitary and undecidable by designing an effective method to reduce Hilbert's tenth problem to unification of extended nominal terms.

In Section 4.3.6, we argued that there is a restricted class of problems for which unification is decidable. Such unification problems contain constraints of form $s \,{}_?{\approx}_?\, t$ where variables are not shared between both terms s, t and variables in t are taken as constants with suspensions. This restricted form of unification constraint is known as matching constraint.

In this chapter, we provide a unification algorithm for the set of matching constraints which it is not only decidable but also complete, by returning the complete set of principal solutions to any unifiable set of matching constraints (and possibly some freshness constraints). Such *matching algorithm* is given in Definition 5.2.6 and its proof of correctness is given in 5.3.18. The appeal of this matching algorithm is that it can be applied as an operational tool to check closedness of extended nominal terms.

We also argued in Section 4.3.6 that, to recover the property of uniqueness of most general solutions it is necessary to restrict the occurrence of a-substitutions suspended over variables on the RHS of ${}_?{\approx}_?$, as a result of a-substitutions being the primary source of non-determinism. It is useful to have a unique most general solution when solving a matching constraint to generate, for instance, rewrite steps.

In this chapter, we begin by describing the notion of *matching problem* and its solution in Section 5.1. Then, the chapter continues by providing a unification algorithm that returns a set of principal solutions to any unifiable matching problem. This is done in Section 5.2, along with some clarifying examples. In Section 5.3, we provide the necessary proofs in order to demonstrate that the unification algorithm is both sound and complete for the class of

matching problems. This is one of the main results of this chapter and can be found in 5.3.17. The other main result is introduced in Section 5.4.2, after applying some syntactic restrictions to both the class of matching problems and the unification algorithm in order to transform it into a unification algorithm for a restricted class of matching problems which enjoys unicity of principal solutions; the proof of correctness can be found in 5.4.11.

We begin the section by providing a definition of the matching problem in the context of extended nominal terms.

5.1 Matching Problems and their Solutions

Generally speaking, a matching constraint, $s \stackrel{?}{\approx} t$, is a form of *one-sided* unification on terms, that is, a unification of terms s,t where only variables in s can be instantiated. We refer to term s as the **pattern** of the matching constraint. Then, variable symbols occurring in t, along with the atom actions suspended over them, are taken as constants. Term t is commonly referred to as the **matched term**. Variable symbols in the pattern are disjoint from variable symbols in the matched term in $s \stackrel{?}{\approx} t$. This notion is formalised below.

Definition 5.1.1. An **extended nominal matching problem** Pr, or **matching problem** for short, is a unification problem as given in Definition 4.1.1 satisfying the condition $(\bigcup_i V(s_i)) \cap (\bigcup_i V(t_i)) = \varnothing$ for all $s_i \stackrel{?}{\approx}_? t_i \in Pr$. Then, write $s_i \stackrel{?}{\approx} t_i$ for each $s_i \stackrel{?}{\approx}_? t_i \in Pr$. A **matching problem-in-context** $(\nabla \vdash s) \stackrel{?}{\approx} (\Delta \vdash t)$ is a unification problem-in-context as in Definition 4.1.2 where $s \stackrel{?}{\approx} t$ and $V(\nabla \vdash s) \cap V(\Delta \vdash t) = \varnothing$.

A **solution** to such problem Pr is a pair $(\mathbf{F}, \sigma)$ of a collection of freshness contexts $\mathbf{F}$ and a v-substitution σ such that $(\mathbf{F}, \sigma)$ is also a solution to the unification problem Pr as given in Definition 4.1.8, with the additional constraint $\mathscr{D}om(\sigma) \subseteq (\bigcup_i V(s_i))$ for all $s_i \stackrel{?}{\approx} t_i \in Pr$. Similarly, a solution to a matching problem-in-context is also a solution $(\mathbf{F}', \sigma')$ to a unification problem-in-context as given in Definition 4.1.9 with the additional constraint $\mathscr{D}om(\sigma') \subseteq V(\nabla \vdash s)$.

Informally, a solution to a matching problem Pr is also a solution to a unification problem with trivial instantiations on the RHS of each matching equation in Pr. Similar to the unification problem, the matching problem also enjoys non-uniqueness of principal solutions for the general case.

In the sequel, we omit the outer set brackets in a singleton collection of sets, $\mathbf{F}$, over freshness contexts.

An example of matching problem and its solution follows.

Example 5.1.2. The set $\{[\mathtt{a} \mapsto \mathtt{Z}]\cdot\mathtt{X} \stackrel{?}{\approx} [\mathtt{c} \mapsto \mathtt{b}]\hat{}(\mathtt{a}\ \mathtt{c})\cdot\mathtt{Y}, \mathtt{c\#X}\}$ is a matching problem whereas the set $\{[\mathtt{a} \mapsto \mathtt{Z}]\cdot\mathtt{X} \stackrel{?}{\approx} [\mathtt{c} \mapsto \mathtt{b}]\hat{}(\mathtt{a}\ \mathtt{c})\cdot\mathtt{Y}, \mathtt{fun}([\mathtt{a}]\mathtt{a}) \stackrel{?}{\approx} \mathtt{Z}\}$ is not, as a result of variable Z occurring on both sides of symbol $\stackrel{?}{\approx}$ in the set.

The pair $(\{\mathtt{c\#Y}\}, [\mathtt{X} \mapsto \mathtt{Y}; \mathtt{Z} \mapsto \mathtt{b}])$ is a solution to the matching problem above since $\mathtt{c\#Y} \vdash ([\mathtt{a} \mapsto \mathtt{Z}]\cdot\mathtt{X})[\mathtt{X} \mapsto \mathtt{Y}; \mathtt{Z} \mapsto \mathtt{b}] \stackrel{?}{\approx} ([\mathtt{c} \mapsto \mathtt{b}]\hat{}(\mathtt{a}\ \mathtt{c})\cdot\mathtt{Y})[\mathtt{X} \mapsto \mathtt{Y}]$ and $\mathtt{c\#Y} \vdash \mathtt{c\#X}[\mathtt{X} \mapsto \mathtt{Y}]$ hold.

We aim to build a matching algorithm for a restricted class of extended terms which enjoys the property of unique most general unifier. This is done incrementally. We begin by defining a sound and complete, but not unitary, matching algorithm for the class of extended terms.

5.2 A Unification Algorithm for Matching Problems

In standard nominal terms, the unification algorithm is also used as a matching tool to check closedness of terms (Fernández et al., 2004). Closed terms and closed rewrite rules are key concepts in higher-order formalisms and instrumental to the translation algorithms in (Domínguez and Fernández, 2014) from NRSs to CRSs and back. This section provides the necessary tools to define a matching algorithm for the class of extended terms. That is, a decidable unification algorithm for matching problems that can be applied to check closedness of extended terms. Correctness of the unification algorithm for matching problems is proven in Theorem 5.3.18. Then, both the unification algorithm and the matching theory is revised in order to provide a unitary unification algorithm for a restricted class of matching problems. Both the unitary unification algorithm and the matching theory are used in Chapter 6 when defining rewriting of extended terms. From now on, we refer to the unification algorithm for matching problems simply as *matching algorithm*. However we continue using predicate symbol $\stackrel{?}{\approx}_?$, along with predicate symbol $\stackrel{?}{\approx}$, since the matching algorithm accepts unification constraints as input.

5.2.1 Auxiliary functions

In order to define a matching algorithm we specify some additional functions to handle matching of terms containing variables as a root symbol.

There are four distinctive unification constraint forms containing variables as a root symbol, namely, $\phi\hat{}\pi\cdot X \stackrel{?}{\approx}_? \phi'\hat{}\pi'\cdot X$, $\phi\hat{}\pi\cdot X \stackrel{?}{\approx}_? \phi'\hat{}\pi'\cdot Y$ and $\phi\hat{}\pi\cdot X \stackrel{?}{\approx}_? t$ and also $t \stackrel{?}{\approx}_? \phi'\hat{}\pi'\cdot X$ for some $X, Y \in \mathscr{X}$ and term t such that $X \neq Y$ and t does not have a variable as root symbol. We have designed a pair of functions to facilitate the matching process when some of the unification constraint forms described above are encountered: function Cap returns

a set of terms used to construct v-substitutions and function Ψ computes the disagreement set of a pair of occurrences of the same variable symbol. More specifically, functions `Cap` and Ψ are applied to any of the constraint forms enumerated above apart from constraints of form $t \;_?{\approx}_?\; \phi'^\frown\pi'{\cdot}X$; this constraint form will be dealt with later, by extending the set of clashing equalities (see Definition 3.2.2) and restricting the generation of v-substitutions to variables occurring originally in pattern terms. We examine both auxiliary functions prior to the definition of the matching algorithm.

Auxiliary function Ψ

Function Ψ originates from the reduction rule case $(\approx_\alpha \mathbf{X})$ in the derivability algorithm given in Definition 3.2.1, used here in the context of unification theory. This is done by replacing predicate $\approx_\alpha$ with predicate $_?{\approx}_?$ as well as changing the view from logical formulæ to set theory. As a result, Ψ is a recursive procedure used to exhaustively search for unification simplification paths that may lead to potential solutions to a unification constraint of form $\phi^\frown\pi{\cdot}X \;_?{\approx}_?\; \phi'^\frown\pi'{\cdot}X$. The recursive definition of Ψ is as follows.

Definition 5.2.1 (Ψ function)**.** For any two occurrences $\phi^\frown\pi{\cdot}X$, $\phi'^\frown\pi'{\cdot}X$ of variable $X \in \mathscr{X}$, domain of atom actions $A = \mathscr{S}upportP(\pi,\pi') \cup \mathscr{D}omP(\phi,\phi')$, and unification problem Pr, $\Psi(\phi^\frown\pi, \phi'^\frown\pi', Pr)^A_X = \{Pr_i \mid i \in \mathrm{I}\}$ where

$$\Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr)^A_X \triangleq \begin{cases} \{Pr\} & \text{, if } A = \varnothing \\ \Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr\cup\{\phi(\pi(a)) \;_?{\approx}_?\; \phi'(\pi'(a))\})^{A\setminus\{a\}}_X \\ \quad\cup\, \Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr\cup\{a\#X\})^{A\setminus\{a\}}_X \text{ (where } a \in A\text{)} & \text{, otherwise} \end{cases}$$

Informally, given a pair of variable occurrences $\phi^\frown\pi{\cdot}X, \phi'^\frown\pi'{\cdot}X$ and unification problem Pr, set $\Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr)^A_X$ generates every ramification of distinct sequences of reduction for unification constraint $\phi^\frown\pi{\cdot}X \;_?{\approx}_?\; \phi'^\frown\pi'{\cdot}X$ which could potentially lead to a favourable solution via application of v-substitutions. The exhaustive generation of unification simplification paths is done as follows. For the base case, that is, when set A is empty, then $\Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr)^\varnothing_X$ contains a singleton set $\{Pr\}$ where Pr is the unification problem provided as argument of function Ψ. Otherwise, A contains the set of atoms in the domain of both atom actions for X and, similarly to core rule $(_?{\approx}_?\mathbf{X})$, we want to test whether $\phi(\pi(a)) \;_?{\approx}_?\; \phi'(\pi'(a))$ or $a\#X$ for each atom $a \in A$. Consequently, there is a pair of corresponding recursive steps $\Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr\cup\{\phi(\pi(a)) \;_?{\approx}_?\; \phi'(\pi'(a))\})^{A\setminus\{a\}}_X$ and $\Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr\cup\{a\#X\})^{A\setminus\{a\}}_X$ which are combined together by means of the set union operator, $\cup$. Note that Ψ can also be applied to matching problems since they are a particular case of the unification problem.

The formation of set $\Psi(\phi^\frown\pi,\phi'^\frown\pi',Pr)^A_X$ is defined recursively on the function arguments $\phi^\frown\pi, \phi'^\frown\pi'$, that is, on the pair of atom actions suspended over each occurrence of variable X.

Each atom action on the pair corresponds to a finite set of mappings suspended over X and each recursive call corresponds to an atom from the finite domain of both atom actions for X, denoted here by A and defined as $\mathscr{S}upportP(\pi,\pi') \cup \mathscr{D}omP(\phi,\phi')$. Hence, we conclude by stating that recursive calls to function Ψ also terminate and it is easy to see that the order in which elements of A are considered is irrelevant since $\Psi(\phi\hat{}\pi, \phi'\hat{}\pi', Pr)^A_X$ is a collection of sets. Therefore, Ψ is indeed a function.

Example 5.2.2 (Function Ψ)**.** By application of Definition 5.2.1 to unification constraint $[\mathtt{a} \mapsto \mathtt{c}; \mathtt{b} \mapsto \mathtt{X}]\cdot\mathtt{X} \;?\approx?\; [\mathtt{a} \mapsto \mathtt{d}; \mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{X}$ we obtain,

$$\begin{array}{l}
\Psi([a \mapsto c; b \mapsto X], [a \mapsto d; b \mapsto a], \varnothing)^{\{a,b\}}_X = \\
\Psi([a \mapsto c; b \mapsto X], [a \mapsto d; b \mapsto a], \{a\#X\})^{\{b\}}_X \cup \\
\Psi([a \mapsto c; b \mapsto X], [a \mapsto d; b \mapsto a], \{c \;?\approx?\; d\})^{\{b\}}_X = \\
\Psi([a \mapsto c; b \mapsto X], [a \mapsto d; b \mapsto a], \{a\#X\} \cup \{b\#X\})^{\varnothing}_X \cup \\
\Psi([a \mapsto c; b \mapsto X], [a \mapsto d; b \mapsto a], \{a\#X\} \cup \{X \;?\approx?\; a\})^{\varnothing}_X \cup \\
\Psi([a \mapsto c; b \mapsto X], [a \mapsto d; b \mapsto a], \{c \;?\approx?\; d\} \cup \{b\#X\})^{\varnothing}_X \\
\Psi([a \mapsto c; b \mapsto X], [a \mapsto d; b \mapsto a], \{c \;?\approx?\; d\} \cup \{X \;?\approx?\; a\})^{\varnothing}_X = \\
\{\{\mathtt{a\#X}, \mathtt{b\#X}\}, \{\mathtt{a\#X}, \mathtt{X} \;?\approx?\; \mathtt{a}\}, \{\mathtt{c} \;?\approx?\; \mathtt{d}, \mathtt{b\#X}\}, \{\mathtt{c} \;?\approx?\; \mathtt{d}, \mathtt{X} \;?\approx?\; \mathtt{a}\}\}.
\end{array}$$

Auxiliary function `Cap`

To solve a unification constraint of form $\phi\hat{}\pi\cdot X \;?\approx?\; t$ where $t \neq \phi'\hat{}\pi'\cdot X$ for some $X \in \mathscr{X}$, one checks if a sub-term of t at some position p, $t|_p$, is contained in the image of ϕ, that is, $[\pi(a) \mapsto s] \in \phi$. In order to find such position p and sub-term s, the matching algorithm generates *cap constraints* $(t[\pi(a)]_p)\phi \;?\approx?\; t$ for every $p \in \mathscr{P}os(t)$ and atom $a \in \mathscr{D}om(\phi)$.

Cap unification is an extension of equational unification widely used in protocol security (see (Anantharaman et al., 2010), for instance): look for a cap to be placed in a given set of terms, so as to unify it with a given term modulo an equational theory. Cap unification of extended nominal terms is assisted by the following auxiliary function.

Definition 5.2.3 (`Cap` terms)**.** Let t be a term, A a set of atoms and $Aux = \{t[a_1 \cdots a_n]_{p_1 \cdots p_m} \mid a_i \in A, p_j \in \mathscr{P}os(t), 1 \leq i \leq n, 1 \leq j \leq m\}$. Then, $\mathtt{Cap}(t,A) = Aux \cup \{t\}$.

Informally, $\mathtt{Cap}(t,A)$ constructs a set of terms that are equivalent to t everywhere except below some positions where replacements from atom set A have been inserted instead. Additionally, $\mathtt{Cap}(t,A)$ also includes term t. A clarifying example follows.

Example 5.2.4. By application of Definition 5.2.3 to term $\mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{T})$ over the signature from Example 2.1.1 and atom set $\{b,c\}$, the following cap terms are generated:

$\mathtt{Cap}(\mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{T}), \{\mathtt{b}, \mathtt{c}\}) =$
$\{\mathtt{b}, \mathtt{c}, \mathtt{cons\ b}, \mathtt{cons\ c}, \mathtt{cons}(\mathtt{b},\mathtt{b}), \mathtt{cons}(\mathtt{b},\mathtt{c}), \mathtt{cons}(\mathtt{c},\mathtt{b}), \mathtt{cons}(\mathtt{c},\mathtt{c}), \mathtt{cons}(\mathtt{b},\mathtt{T}),$
$\mathtt{cons}(\mathtt{c},\mathtt{T}), \mathtt{cons}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{F}, \mathtt{b}), \mathtt{cons}([\mathtt{a} \mapsto \mathtt{c}]\cdot\mathtt{F}, \mathtt{b}), \mathtt{cons}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{F}, \mathtt{c}),$
$\mathtt{cons}([\mathtt{a} \mapsto \mathtt{c}]\cdot\mathtt{F}, \mathtt{c}), \mathtt{cons}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{F}, \mathtt{T}), \mathtt{cons}([\mathtt{a} \mapsto \mathtt{c}]\cdot\mathtt{F}, \mathtt{T}), \mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{T})\}.$

5.2.2 A matching algorithm for extended terms

The set of clashing equalities given in Definition 3.2.2 must be updated to include matching constraints which cannot be solved because of the restrictions imposed on matching problems, namely, variable symbols on matched terms are no longer instantiated. Then, the updated set of clashing equalities reduces to $\varnothing$ with rule $({}_?{\approx}_?\bot)$. We clarify further after the definition.

Definition 5.2.5 (Clashing equalities in matching problems)**.** The set of clashing equalities contains matching constraints of form $s \ {}_?{\approx}\ t$ where s,t have different term constructors at the root, different atoms, different moderated variables or function applications with different term formers. There is one exception to the definition: if $s \ {}_?{\approx}\ t \in Pr$ and $s = \phi\hat{}\pi\cdot X$ where $X \notin V_{RHS}(Pr)$ for some matching problem Pr, then $\phi\hat{}\pi\cdot X \ {}_?{\approx}\ t$ *is not* a clashing equality.

The set of clashing equalities is similar to the set previously described in Definition 3.2.2 to check α-equality, apart from the given exception. That is, for any given problem Pr, constraints of form $\phi\hat{}\pi\cdot X \ {}_?{\approx}\ t \in Pr$ are not clashing equalities if either X does not occur on any matched term in Pr or t has form $\phi'\hat{}\pi'\cdot X$. The former is not a clashing equality because variables on the LHS of matching problems will generate non-trivial v-substitutions in order to unify both sides of the problem (recall terms on the RHS of Pr are taken as constants) whereas the latter triggers a matching rule based on $({\approx}_\alpha\mathbf{X})$ from Definition 3.2.1 to search for the disagreement set between both variable terms and the primitive constraints that entail the derivation of such constraint with respect to the atoms in the disagreement set. The need to differentiate variable symbols occurring in matched terms from those occurring on pattern terms leads to the inclusion of an additional parameter in the matching algorithm given in Definition 5.2.6 to check whether a variable symbol belongs to either the matched or the pattern term. That is, there is some *pre-cooking* before applying the matching algorithm; $V_{RHS}(Pr)$ (see Definition 3.1.4) is obtained before application of the matching rules below to syntactically restrict instantiation to those variables that do no occur in $V_{RHS}(Pr)$ since variables on the RHS of a unification constraint are taken as constants.

We are now ready to present a reduction system for the derivation of principal solutions to matching problems. The algorithm is essentially the one described by Urban et al (Urban et al., 2004), generalised to include suspended a-substitutions on variables. It follows the

notation given in (Baader and Snyder, 2001), which in turn is based on the seminal work on first-order unification by Martelli and Montanari (Martelli and Montanari, 1982).

Definition 5.2.6 (Matching steps)**.** Let Pr, Pr', and θ, θ' be a pair of matching problems and v-substitutions respectively and fix $Xs = V_{RHS}(Pr_0)$ where Pr_0 is the input matching problem. Below, s, t denote terms, a, b distinct atoms, X, Y distinct variables, f a term former. Additionally, $\cup$ denotes set union of pairs of form (Pr, θ) (we omit set brackets) and Ψ, $\mathtt{Cap}$ are the functions defined in Definitions 5.2.1 & 5.2.3 respectively.

Then, $(Xs, (Pr, \theta) \Longrightarrow_{?\approx} (Pr', \theta'))$, written $(Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} (Pr', \theta')$, is a pair consisting of set Xs and the one-step matching reduction relation defined as follows.

$$(?\approx\equiv) \quad (\{t \mathrel{?\approx?} t\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} (Pr, \theta)$$

$$(?\approx[\mathbf{a}]) \quad (\{[a]s \mathrel{?\approx?} [a]t\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} (\{s \mathrel{?\approx?} t\} \cup Pr, \theta)$$

$$(?\approx[\mathbf{b}]) \quad (\{[a]s \mathrel{?\approx?} [b]t\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} (\{(b\ a)\cdot s \mathrel{?\approx?} t, b\#s\} \cup Pr, \theta)$$

$$(?\approx\mathbf{f}) \quad (\{fs \mathrel{?\approx?} ft\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} (\{s \mathrel{?\approx?} t\} \cup Pr, \theta)$$

$$(?\approx\mathbf{tupl}) \quad (\{(s_1, \ldots, s_n) \mathrel{?\approx?} (t_1 \ldots, t_n)\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} (\{s_1 \mathrel{?\approx?} t_1, \ldots, s_n \mathrel{?\approx?} t_n\} \cup Pr, \theta)$$

$$(?\approx\mathbf{X})^1 \quad (\{\phi\hat{}\pi\cdot X \mathrel{?\approx?} \phi'\hat{}\pi'\cdot X\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} \bigcup_{Pr' \in \Psi(\phi\hat{}\pi, \phi'\hat{}\pi', \varnothing)^A_X} \{(Pr' \cup Pr, \theta)\}$$

$$(?\approx\mathbf{Inst})^1 \quad (\{\phi\hat{}\pi\cdot X \mathrel{?\approx?} t\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} \bigcup_{s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))} \{(\{s(\phi\theta') \mathrel{?\approx?} t\} \cup Pr\theta', \theta \bullet \theta')\}$$

(where $t \neq \phi'\hat{}\pi'\cdot Y$ for some $Y \in \mathscr{X}$, $(X \notin Xs)$ and $\theta' = [X \mapsto \pi^{-1}\cdot s]$)

$$(?\approx\mathbf{XY})^1 \quad (\{\phi\hat{}\pi\cdot X \mathrel{?\approx?} \phi'\hat{}\pi'\cdot Y\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} \bigcup_{s \in (\mathtt{Cap}(\phi'\hat{}\pi'\cdot Y, \mathscr{D}om(\phi)) \cup \{\pi'\cdot Y\})} \{(\{s(\phi\theta') \mathrel{?\approx?} \phi'\hat{}\pi'\cdot Y\} \cup Pr\theta', \theta \bullet \theta')\}$$

(where $(X \notin Xs)$ and $\theta' = [X \mapsto \pi^{-1}\cdot s]$)

$$(?\approx\bot)^1 \quad (\{s \mathrel{?\approx?} t\} \cup Pr, \theta) \overset{Xs}{\Longrightarrow}_{?\approx} \varnothing \text{ (where } s \mathrel{?\approx?} t \text{ are clashing)}$$

For the set of matching rules, rule $(?\approx\equiv)$ is applied first; the rest of the matching rules are described in no particular order. Matching rules, along with the set of freshness rules from Definition 3.3.6, define a reduction relation $\Longrightarrow$ on matching problems Pr.

Let $\mathbb{P}, \mathbb{Q}$ denote sets of pairs of form (Pr, θ). Then, for any given matching problem Pr_0 with $Xs = V_{RHS}(Pr_0)$, write $\mathbb{P} \overset{Xs}{\Longrightarrow}_{?\approx} \mathbb{Q}$ (resp. $\mathbb{P} \Longrightarrow_{\#} \mathbb{Q}$), if $\mathbb{Q}$ is obtained from $\mathbb{P}$ by application of one matching (resp. freshness) reduction rule. As usual, $\Longrightarrow^*_{?\approx}$ (resp. $\Longrightarrow^*_{\#}$) denotes reflexive transitive closure. Arrow subindices are omitted if it does not matter to which subset of rules the step belongs. Then, write $\Longrightarrow_*$ when generalising. Similarly,

[1] In this rule, the right-hand side is a *set*; we assume a flattening step is performed after each application of the rule.

Xs is commonly omitted from any one-step matching reduction rule when unambiguous, instead set membership notation is made explicit for rules ($_?\approx$**Inst**) and ($_?\approx$**XY**). That is, write $\mathbb{P} \overset{X\in Xs}{\Longrightarrow}_\Omega \mathbb{Q}$ or $\mathbb{P} \overset{X\notin Xs}{\Longrightarrow}_\Omega \mathbb{Q}$ where Ω is rule ($_?\approx$**Inst**) or rule ($_?\approx$**XY**).

We have imposed an order on the reduction of constraints such that unification constraints have a higher priority over freshness constraints. Then, the *two-phase strategy matching process* is defined as follows:

Phase 1:

$$(V_{RHS}(Pr_0), \{(Pr_0, \mathtt{Id})\} \Longrightarrow^*_{?\approx} \langle Pr_0\rangle_{\mathtt{nf}_{?\approx}})$$

Phase 2:

$$\text{for each } (Pr_i, \theta_i) \in \langle Pr_0\rangle_{\mathtt{nf}_{?\approx}} : \begin{cases} (\{Pr_1\}, \theta_1) & \Longrightarrow^*_{\#} (\langle Pr_1\rangle_{\mathtt{nf}}, \theta_1) \\ \quad\vdots & \\ (\{Pr_n\}, \theta_n) & \Longrightarrow^*_{\#} (\langle Pr_n\rangle_{\mathtt{nf}}, \theta_n) \end{cases}$$

Solution set:

$$W' = \{(\langle Pr_i\rangle_{\mathtt{nf}}, \theta_i) \mid (Pr_i, \theta_i) \in \langle Pr_0\rangle_{\mathtt{nf}_{?\approx}}, \langle Pr_i\rangle_{\mathtt{nf}} \neq \bot\}$$

where $1 \leq i \leq n$

where Pr_0 is the input matching problem, $\langle Pr_i\rangle_{\mathtt{nf}}$ is the normal form of a set of freshness constraints Pr_i as defined in Lemma 4.1.3 and $\langle Pr_0\rangle_{\mathtt{nf}_{?\approx}}$ is the normal form of Pr_0 by application of the set of matching reduction rules above.

In the sequel, matching rules ($_?\approx$**Inst**) and ($_?\approx$**XY**) are also known as **instantiating rules**, whereas every other matching rule is referred to as a **non-instantiating rule**.

Given a matching problem Pr_0, matching rule ($_?\approx\bot$) removes from the matching process any pair of form $(\{s \,{}_?\!\approx_?\, t\} \cup Pr, \theta)$ which has not pattern-matched with any other matching rule above, that is, $s \,{}_?\!\approx_?\, t$ is any clashing equality as in Definition 5.2.5. Recall that set $V_{RHS}(Pr_0)$ is a parameter used in the matching process to discard constraints of form $\phi\hat{}\pi\cdot X \,{}_?\!\approx_?\, t$ whenever $X \in V_{RHS}(Pr_0)$ and $t \neq \phi'\hat{}\pi'\cdot X$. Additionally, observe that any pair (Pr_i, θ_i) in $\langle Pr\rangle_{\mathtt{nf}_{?\approx}}$ is discarded from the solution set whenever $\langle Pr_i\rangle_{\mathtt{nf}} = \bot$. Therefore, an unsolvable matching problem is represented by the empty set.

As a result of Definition 5.2.3, trivial constraints may occur after application of rule ($_?\approx$**Inst**) or rule ($_?\approx$**XY**), that is, constraints with syntactically equal terms at both sides. Instead of simplifying a trivial constraint step by step, we have provided an optimisation on the number of unification reductions by discarding trivial constraints of form $t \,{}_?\!\approx_?\, t$. This

is defined in rule $(_?{\approx}{\equiv})$ which subsumes the previous rule for atom equivalence $(_?{\approx}\mathbf{a})$ and therefore omitted.

Note that rule $(_?{\approx}\mathbf{XY})$ is a particular case of rule $(_?{\approx}\mathbf{Inst})$, extended with an additional branch of reduction that involves replacing $\pi{\cdot}X$ by $\pi'{\cdot}Y$ in a unification constraint of form $\phi\hat{}\pi{\cdot}X \;_?{\approx}_?\; \phi'\hat{}\pi'{\cdot}Y$. As a result, $\pi'{\cdot}Y$ has been added to set $\mathtt{Cap}(\phi'\hat{}\pi'{\cdot}Y, \mathscr{D}om(\phi))$ in rule $(_?{\approx}\mathbf{XY})$. Then, a successful solution is always obtained by finding the difference set between $\phi[X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)]$ and ϕ'. An alternative approach would have been to subsume rule $(_?{\approx}\mathbf{XY})$ into rule $(_?{\approx}\mathbf{Inst})$. This is done by discarding side condition $t \neq \phi'\hat{}\pi'{\cdot}Y$ from rule $(_?{\approx}\mathbf{Inst})$ and extending Definition 5.2.3 to include $\pi'{\cdot}Y$ in set $\mathtt{Cap}(t, \mathscr{D}om(\phi))$ whenever $t = \phi'\hat{}\pi'{\cdot}Y$. However, since $\pi'{\cdot}Y$ is not a subterm of $\phi'\hat{}\pi'{\cdot}Y$ following Definition 2.1.13, we made the design choice of not include $\pi'{\cdot}Y$ as part of the definition of $\mathtt{Cap}$.

Given a matching problem Pr, the matching process proceeds in two phases, it begins by applying matching rules, $\Longrightarrow_{_?{\approx}}$, non-deterministically until no equations of form $s \;_?{\approx}_?\; t$ are left, resolving into a (possibly empty) set of pairs $\{(Pr_1, \theta_1), \ldots, (Pr_m, \theta_m)\}$ where each Pr_i is a set of freshness constraints and each θ_i a composition of v-substitutions. Next, during the second phase of the algorithm, function $\langle\cdot\rangle_{\mathtt{nf}}$ from Lemma 4.1.3 is applied to each Pr_i, resulting in a collection of consistent freshness contexts $\{\nabla_{i1}, \ldots, \nabla_{ik}\} = \langle Pr_i\rangle_{\mathtt{nf}}$ or otherwise $\langle Pr_i\rangle_{\mathtt{nf}} = \bot$. Finally, the solution set is constructed with each pair $(\langle Pr_i\rangle_{\mathtt{nf}}, \theta_i)$ where $\langle Pr_i\rangle_{\mathtt{nf}} \neq \bot$.

The two-phase strategy has been chosen in order to avoid extra complexity during the reduction sequence, as a result of unification constraints and freshness constraints having distinct normal forms, namely, a set and a collection of sets respectively. However, this is a design choice. We could have introduced a binary operator to deal with the union of a constraint of form $a\#s$ and the remaining unification problem Pr when a normal form is reached for $a\#s$, $\langle a\#s\rangle_{\mathtt{nf}}$. Then, assuming $\langle a\#s\rangle_{\mathtt{nf}} \neq \bot$, $\bigcup_{\Delta_i \in \langle a\#s\rangle_{\mathtt{nf}}} (\Delta_i \cup Pr, \theta)$ would represent the set of solution candidates to $\{a\#s\} \cup Pr$, with θ the v-substitution generated during the reduction sequence to $\{a\#s\} \cup Pr$. However, note that this approach generates repetition of the reduction sequence for Pr for each candidate solution $(\Delta_i \cup Pr, \theta)$.

By systematically enumerating all possible candidates for the solution of a matching problem, the two-phase matching algorithm may produce pairs of solutions, $(\mathbf{F}, \sigma), (\mathbf{F}', \sigma')$, sharing common unifiers, that is, $\nabla \vdash \sigma \bullet \mathtt{Id} \approx_\alpha \sigma'$ for each $\nabla \in \mathbf{F}'$, following the notation given in the first bullet point from Definition 4.1.14. Take for instance the pair of solutions $(\{\mathtt{a\#Z}\}, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto \mathtt{Z}])$, $(\varnothing, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]{\cdot}\mathtt{Z}])$ from solution set W in Example 5.2.10. Both pairs have α-equivalent unifiers with respect to collection of freshness contexts $\{\mathtt{a\#Z}\}$ since $\mathtt{a\#Z} \vdash [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto \mathtt{Z}] \bullet \mathtt{Id} \approx_\alpha [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]{\cdot}\mathtt{Z}]$ but
$\nvdash [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]{\cdot}\mathtt{Z}] \bullet \mathtt{Id} \approx_\alpha [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto \mathtt{Z}]$ thus the solution set W benefits from mer-

ging both solution pairs into a unique principal solution $(\{\{\mathtt{a\#Z}\},\varnothing\},[\mathtt{X}\mapsto\mathtt{a};\mathtt{Y}\mapsto[\mathtt{a}\mapsto\mathtt{b}]\cdot\mathtt{Z}])$ which preserves the semantics of both solutions in W. Another case is solutions sharing a unifier but differing on the collection of freshness contexts, being one of them the empty set. take for instance the pair of solutions $(\varnothing,[\mathtt{X}\mapsto(\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$ and $(\{\mathtt{a\#Y},\mathtt{b\#Y},\mathtt{c\#Y}\},[\mathtt{X}\mapsto(\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$ from the solution set in Example 5.2.8. If a v-substitution is a unifier under empty assumptions, then any other freshness context for such unifier is valid yet redundant. As a result, solution $(\{\mathtt{a\#Y},\mathtt{b\#Y},\mathtt{c\#Y}\},[\mathtt{X}\mapsto(\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$ can be safely discarded from the solution set.

Nevertheless, since matching reduction rules from Definition 5.2.6 and freshness reduction rules from Definition 3.3.6 are the core of the matching algorithm, we have decided to verify the properties of termination and confluence prior to the optimisation of the results by merging solutions with common unifiers. The merging procedure can be found after, in Definition 5.3.4 followed by a couple of examples in Example 5.3.5 and Example 5.3.6.

Next, we provide some examples of application of the matching algorithm.

5.2.3 Examples of application of the matching algorithm

Below, a pair of examples illustrate application of the matching algorithm for both matching problems and matching problems-in-context. We have labelled each one-step reduction with the rule names for readability.

Example 5.2.7 (Matching problem-in-context)**.** By application of the matching algorithm, the matching of terms-in-context $\mathtt{a\#X}\vdash\mathtt{X}$ and $\mathtt{a}$ is unsolvable, as it is shown below.
Phase 1:
$(\{\mathtt{X}\ {}_?{\approx}_?\ \mathtt{a},\mathtt{a\#X}\},\mathtt{Id})\overset{x\notin\varnothing}{\Longrightarrow}_{({}_?\approx\textbf{Inst})}(\{\mathtt{a}\ {}_?{\approx}_?\ \mathtt{a},\mathtt{a\#a}\},\mathtt{Id}\bullet[\mathtt{X}\mapsto\mathtt{a}])$
(where $\mathtt{Cap}(a,\varnothing)=\{a\}$)
$\Longrightarrow_{({}_?\approx\equiv)}(\{\mathtt{a\#a}\},\mathtt{Id}\bullet[\mathtt{X}\mapsto\mathtt{a}])$
Phase 2:
$\Longrightarrow_{(\#\bot)}(\{\bot\},[\mathtt{X}\mapsto\mathtt{a}])$.
Solution set: $\varnothing$ since $\langle\{\mathtt{a\#a}\}\rangle_{\mathtt{nf}}=\bot$.

Example 5.2.8 (Matching problem I)**.** The matching problem $\{(\mathtt{a}\ \mathtt{b})\cdot\mathtt{X} \stackrel{?}{\approx} (\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}\}$ has solution set

$(\{\mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}, \mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{b\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{b\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\varnothing, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}, \mathtt{b\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{a\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{b\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{b\#Y}, \mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$(\{\mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$

by application of the matching algorithm as follows.
$(\{(\mathtt{a}\ \mathtt{b})\cdot\mathtt{X} \stackrel{?}{\approx}_? (\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}\}, \mathtt{Id})$
$\stackrel{X\notin\{Y\}}{\Longrightarrow}_{(?\approx\mathbf{XY})} (\{(\mathtt{a}\ \mathtt{b})(\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y} \stackrel{?}{\approx}_? (\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}\}, \mathtt{Id} \bullet [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
(where $\mathtt{Cap}((\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}, \varnothing) = \{(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}\}$)
$\Longrightarrow_{(?\approx\mathbf{X})} (\{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$\cup (\{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\{\mathtt{a\#Y}, \mathtt{b\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$\cup (\{\mathtt{a\#Y}, \mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\{\mathtt{a\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$\cup (\{\mathtt{a\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\{\mathtt{a\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$\cup (\{\mathtt{b\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\{\mathtt{b\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$\cup (\{\mathtt{b\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\{\mathtt{b\#Y}, \mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$\cup (\{\mathtt{c\#Y}, \mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\{\mathtt{c\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
$\cup (\{\mathtt{d\#Y}\}, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}]) \cup (\varnothing, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])$
(where $\Psi((\mathtt{a}\ \mathtt{b})(\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d}), (\mathtt{c}\ \mathtt{d}), \varnothing)_{\mathtt{Y}}^{\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d}\}} = \{\varnothing, \{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}$
$\{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{c\#Y}\}, \{\mathtt{a\#Y}, \mathtt{b\#Y}, \mathtt{d\#Y}\}, \{\mathtt{a\#Y}, \mathtt{b\#Y}\}, \{\mathtt{a\#Y}, \mathtt{c\#Y}\},$
$\{\mathtt{a\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, \{\mathtt{a\#Y}, \mathtt{d\#Y}\}, \{\mathtt{a\#Y}\}, \{\mathtt{b\#Y}, \mathtt{d\#Y}\}, \{\mathtt{b\#Y}\},$
$\{\mathtt{b\#Y}, \mathtt{c\#Y}, \mathtt{d\#Y}\}, \{\mathtt{b\#Y}, \mathtt{c\#Y}\}, \{\mathtt{c\#Y}, \mathtt{d\#Y}\}, \{\mathtt{c\#Y}\}, \{\mathtt{d\#Y}\}\}$.

In Example 5.3.6 we show how the solution set above benefits from the merging of solutions procedure given in Definition 5.3.4. Then, the solution set above is transformed into a singleton.

Example 5.2.9 (Matching problem II)**.** The matching problem
$\{([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}, [\mathtt{a}]\mathtt{X}) \stackrel{?}{\approx} ([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{M}, [\mathtt{a}]\mathtt{Z})\}$ is unsolvable as shown by application of the matching algorithm.
$(\{([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}, [\mathtt{a}]\mathtt{X}) \stackrel{?}{\approx}_? ([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{M}, [\mathtt{a}]\mathtt{Z})\}, \mathtt{Id})$
$\Longrightarrow_{(?\approx\mathbf{tupl})} (\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{M}, [\mathtt{a}]\mathtt{X} \stackrel{?}{\approx}_? [\mathtt{a}]\mathtt{Z}\}, \mathtt{Id})$

$\Longrightarrow_{(?\approx[\mathbf{a}])} (\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{M}, \mathtt{X} \stackrel{?}{\approx}_? \mathtt{Z}\}, \mathtt{Id})$
$\stackrel{\{\mathtt{M},\mathtt{Z}\}}{\Longrightarrow}_{(?\approx\mathbf{Inst})} (\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{M}\}, \mathtt{Id} \bullet \theta)$
where $\mathtt{X} \notin \{\mathtt{M}, \mathtt{Z}\}$, $\mathtt{Cap}(\mathtt{Z}, \varnothing) = \mathtt{Z}$ such that $\theta = [\mathtt{X} \mapsto \mathtt{Z}]$ and $\vdash ([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X})\theta \approx_\alpha [\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{Z}$
$\stackrel{\{\mathtt{M},\mathtt{Z}\}}{\Longrightarrow}_{(?\approx\perp)} \varnothing$ since $\mathtt{Z} \in \{\mathtt{M}, \mathtt{Z}\}$.

Any matching constraint containing some variable as root symbol which does not conform to the specifications of rule $(?\approx\mathbf{X})$, rule $(?\approx\mathbf{Inst})$ or rule $(?\approx\mathbf{XY}_?)$ is now reduced to $\varnothing$ by application of rule $(?\approx\perp)$ as described in Definition 5.2.5.

Example 5.2.10 (Matching problem III)**.** The unification problem given in Example 4.1.12 is also a matching problem, and it has the following set of solutions W,

$$(\varnothing, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}]) \quad (\varnothing, [\mathtt{X} \mapsto \mathtt{Z}; \mathtt{Y} \mapsto \mathtt{b}]) \qquad (\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])$$
$$(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}]) \qquad\qquad (\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto \mathtt{Z}])$$

by application of the matching algorithm as follows.
$(\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, \mathtt{Id}) \stackrel{X\notin\{Z\}}{\Longrightarrow}_{(?\approx\mathbf{XY})} \{(\{\mathtt{Y} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, \mathtt{Id} \bullet [\mathtt{X} \mapsto \mathtt{a}])\}$
$\quad \cup\{(\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, \mathtt{Id} \bullet [\mathtt{X} \mapsto \mathtt{Z}])\}$
$\quad \cup\{(\{[\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, \mathtt{Id} \bullet [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
(where $\mathtt{Cap}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}, \{\mathtt{a}\}) = \{\mathtt{a}, \mathtt{Z}, [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}$)
$\Longrightarrow_{(?\approx\equiv)} \{(\{\mathtt{Y} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}])\} \cup\{(\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])\}$
$\quad \cup\{(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
$\stackrel{Y\notin\{Z\}}{\Longrightarrow}_{(?\approx\mathbf{XY})} \{(\{[\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
$\quad \cup\{(\{\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto \mathtt{Z}])\}$
$\quad \cup\{(\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])\} \cup\{(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
(where $\mathtt{Cap}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}, \varnothing) = \{[\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}$)
$\Longrightarrow_{(?\approx\mathbf{X})} \{(\{[\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
$\quad \cup\{(\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto \mathtt{Z}])\}$
$\quad \cup\{(\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])\} \cup\{(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
(where $\Psi(\mathtt{Id}, [\mathtt{a} \mapsto \mathtt{b}])^{\{\mathtt{a}\}}_{\mathtt{Z}} = \{\{\mathtt{a}\#\mathtt{Z}\}\}$
$\Longrightarrow_{(?\approx\equiv)} \{(\varnothing, [X \mapsto a] \bullet [Y \mapsto [a \mapsto b]\cdot Z])\} \cup\{(\{[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{Z} \stackrel{?}{\approx}_? [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])\}$
$\quad \cup\{(\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto \mathtt{Z}])\} \cup\{(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
$\Longrightarrow_{(?\approx\mathbf{X})} \{(\varnothing, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\} \cup\{(\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto \mathtt{Z}])\}$
$\quad \cup\{(\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])\} \cup\{(\{\mathtt{Y} \stackrel{?}{\approx}_? \mathtt{b}\}, [\mathtt{X} \mapsto \mathtt{Z}])\} \cup\{(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$
(where $\Psi([\mathtt{a} \mapsto \mathtt{Y}], [\mathtt{a} \mapsto \mathtt{b}], \varnothing)^{\{\mathtt{a}\}}_{\mathtt{Z}} = \{\{\mathtt{a}\#\mathtt{Z}\}, \{\mathtt{Y} \stackrel{?}{\approx}_? \mathtt{b}\}\}$)
$\stackrel{Y\notin\{Z\}}{\Longrightarrow}_{(?\approx\mathbf{Inst})} \{(\varnothing, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\} \cup\{(\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto \mathtt{Z}])\}$
$\quad \cup\{(\{\mathtt{a}\#\mathtt{Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])\} \cup\{(\{\mathtt{b} \stackrel{?}{\approx}_? \mathtt{b}\}, [\mathtt{X} \mapsto \mathtt{Z}] \bullet [\mathtt{Y} \mapsto \mathtt{b}])\}$

$$\cup\{(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}$$
$$\Longrightarrow_{(?\approx\equiv)} \{(\varnothing, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\} \cup\{(\{\mathtt{a\#Z}\}, [\mathtt{X} \mapsto \mathtt{a}] \bullet [\mathtt{Y} \mapsto \mathtt{Z}])\}$$
$$\cup\{(\{\mathtt{a\#Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])\} \cup\{(\varnothing, [\mathtt{X} \mapsto \mathtt{Z}] \bullet [\mathtt{Y} \mapsto \mathtt{b}])\} \cup\{(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])\}.$$

5.3 Properties of the Matching Algorithm

In this section we show that the matching algorithm is convergent, that is, terminating and confluent. Then, the matching algorithm can be represented as a function from matching problems to their unique set of solutions. Finally, we show that the matching algorithm is correct, that is, that for any unifiable matching problem, it returns the complete set of principal solutions to such problem, otherwise, it returns the empty set.

5.3.1 Termination, confluence and normal form of matching problems

We are ready to prove the matching algorithm terminates for any arbitrary matching problem. The termination proof is demonstrated following the two-phase strategy matching process stated in Definition 5.2.6.

Termination of non-instantiating cases has already been proven for the constraint checking algorithm in Lemma 3.2.5. The instantiating cases also terminate. To demonstrate such result we define a complexity measure $\langle n_1, n_2, n_3\rangle$ for instantiating terms $s \;{}_?{\approx}_?\; t$ where n_1 is the number of distinct variables in (s,t), n_2 is the size of $s \;{}_?{\approx}_?\; t$, that is $|s| + |t|$ (see Definition 2.1.13) and n_3 is the weight of the predicate where we assigned 2 to predicate ${}_?{\approx}_?$ and 1 to predicate #.

Lemma 5.3.1 (Termination)**.** *For any finite matching problem Pr, each sequence of transformations*

$$\{(Pr, \mathtt{Id})\} \overset{Xs}{\Longrightarrow}_{?\approx} \begin{cases} (Pr_{11}, \theta_{11}) \overset{Xs}{\Longrightarrow}_{?\approx} \ldots \overset{Xs}{\Longrightarrow}_{?\approx} (Pr_{1k}, \theta_{1k}) \Longrightarrow_{\#} (Pr_{1(k+1)}, \theta_{1k}) \Longrightarrow_{\#} \ldots \\ \quad \vdots \\ (Pr_{n1}, \theta_{n1}) \overset{Xs}{\Longrightarrow}_{?\approx} \ldots \overset{Xs}{\Longrightarrow}_{?\approx} (Pr_{nm}, \theta_{nm}) \Longrightarrow_{\#} (Pr_{n(m+1)}, \theta_{nm}) \Longrightarrow_{\#} \ldots \end{cases}$$

terminates, either with $(\{\bot\}, \theta)$ and the pair is discarded from the set of solutions, or with $(\mathbf{F}, \theta)$ where $\mathbf{F}$ is a collection of freshness contexts and θ a composition of v-substitutions. Above, $Xs = V_{RHS}(Pr)$.

Proof. See Appendix B. ■

Having proved that the matching algorithm is a terminating system, we continue by showing that it is also confluent under the specific strategy imposed to the matching algorithm. This is the case since matching rules have no non-trivial overlaps.

Lemma 5.3.2 (Confluence)**.** *The relation $\Longrightarrow$ is confluent such that, if there exists a pair of reduction steps $(Pr,\theta) \Longrightarrow_* (Pr_1,\theta_1)$ and $(Pr,\theta) \Longrightarrow_* (Pr_2,\theta_2)$, then there also exists (Pr_3,θ_3) such that $(Pr_1,\theta_1) \Longrightarrow_* (Pr_3,\theta_3)$ and $(Pr_2,\theta_2) \Longrightarrow_* (Pr_3,\theta_3)$.*

Proof. See Appendix B. ■

As a consequence of the termination and confluence properties, the reduction relation $\Longrightarrow$ defines a function from matching problems to their unique normal forms.

Definition 5.3.3. Write $\texttt{Match}(Pr)$ for the normal form of matching problem Pr following the two-phased strategy described in Definition 5.2.6.

Then, $\langle Pr \rangle_{sol}$ is the function that merges solutions from $\texttt{Match}(Pr)$ with α-equivalent unifiers by application of the following definition.

Definition 5.3.4 (Merging solutions)**.** Let W be a set of solutions, such that there are two different elements $(\mathbf{F},\sigma)$ and $(\mathbf{F}',\sigma')$ in W satisfying $\forall \Delta \in \mathbf{F}.\Delta \vdash \sigma \approx_\alpha \sigma'$. Then, the pair of solutions can be replaced with a single solution as follows:

$$([\mathbf{W_1}]) \quad (\mathbf{F},\sigma),(\mathbf{F}',\sigma') \quad \Longrightarrow_{[W]} \quad (\mathbf{F}' \cup \mathbf{F},\sigma')$$

Further, if solution $(\mathbf{F},\sigma)$ contains the empty set as one of the freshness contexts in $\mathbf{F}$ and $\mathbf{F}$ is not a singleton, then any other freshness context in $\mathbf{F}$ is redundant and can be discarded as follows:

$$([\mathbf{W_2}]) \quad (\mathbf{F},\sigma) \quad \Longrightarrow_{[W]} \quad (\{\varnothing\},\sigma) \quad (\text{where } \mathbf{F} \neq \{\varnothing\} \text{ and } \varnothing \in \mathbf{F})$$

Write $[W]$ for the normal form of W by the rules above.

Informally, given a set of solutions W to some unifiable matching problem Pr, for the case of rule $([\mathbf{W_1}])$, the definition above checks solutions in W pair-wise, $(\mathbf{F},\sigma),(\mathbf{F}',\sigma') \in W$, to find common unifiers modulo α which are derivable from a collection of freshness contexts occurring in one of the solutions, i.e., $\Delta \vdash \sigma \approx_\alpha \sigma'$ for each $\Delta \in \mathbf{F}$. Such a pair of solutions is replaced by $(\mathbf{F}' \cup \mathbf{F},\sigma')$. The newly generated solution preserves the semantics of the original pair of solutions. Additionally, for the case of rule $([\mathbf{W_2}])$, the definition above discards redundant freshness contexts $\Delta \in \mathbf{F}$ for any solution $(\mathbf{F},\sigma)$ to Pr such that $\mathbf{F}$ is not a singleton containing $\varnothing$ yet $\varnothing$ is one of the freshness contexts in $\mathbf{F}$. Then, observe that $\varnothing$ subsumes any

other freshness context by the property of weakening (see Lemma 2.5.1) and thus the result can be strengthened by discarding every $\Delta \in \mathbf{F}$ distinct to $\varnothing$.

In Section 5.4.2, we show how the merging of solutions is instrumental for the matching algorithm to provide unique most general solutions for a set of restricted matching problems. Next, some clarifying examples.

Example 5.3.5. By application of $[\cdot]$ to the set of solutions W from Example 5.2.10 we have generated solution $(\varnothing, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])$ replacing in W the pair of solutions $(\varnothing, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])$, $(\{\mathtt{a\#Z}\}, [\mathtt{X} \mapsto \mathtt{a}; \mathtt{Y} \mapsto \mathtt{Z}])$ by application of rule $[W_1]$ and subsequent application of rule $[W_2]$. Similarly, generated solution $(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])$ replaces the pair of solutions $(\varnothing, [\mathtt{X} \mapsto [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Z}])$, $(\{\mathtt{a\#Z}\}, [\mathtt{X} \mapsto \mathtt{Z}])$ from W.

Example 5.3.6. By application of $[\cdot]$ to the set of solutions from Example 5.2.8, we obtain the singleton solution set $\{(\varnothing, [\mathtt{X} \mapsto (\mathtt{a}\ \mathtt{b})(\mathtt{c}\ \mathtt{d})\cdot\mathtt{Y}])\}$.

Now, after proving that the set of rules is convergent (see Lemma 5.3.1 and Lemma 5.3.2), we are ready to characterise the normal form produced by the matching algorithm and filtered by Definition 5.3.4.

Lemma 5.3.7 (Representation of normal forms)**.**

- $\langle\{a\#s\}\rangle_{sol} = [\mathtt{Match}(\{a\#s\}, \varnothing)] = \{(\langle\{a\#s\}\rangle_{\mathtt{nf}}, \mathtt{Id})\}$ *if* $\langle\{a\#s\}\rangle_{\mathtt{nf}} \neq \bot$, *where* $\langle\{a\#s\}\rangle_{\mathtt{nf}}$ *is a collection of freshness contexts or otherwise,* $\langle a\#s\rangle_{sol} = \varnothing$*;*

- $\langle\{s \stackrel{?}{\approx}_? t\}\rangle_{sol} = [\mathtt{Match}(\{s \stackrel{?}{\approx}_? t\}, V_{RHS}(\{s \stackrel{?}{\approx}_? t\}))] = \{(\mathbf{F}_i, \theta_i) \mid i \in I\}$ *where each* $\mathbf{F}_i$ *is a collection of freshness contexts and each* θ_i *is a distinct unifier or otherwise,* $\langle\{s \stackrel{?}{\approx}_? t\}\rangle_{sol} = \varnothing$*;*

- *An immediate consequence is that* $\langle Pr\rangle_{sol}$ *has also form* $\{(\mathbf{F}_i, \theta_i) \mid i \in I\}$ *as above, or otherwise* $\varnothing$*, if* $\langle C\rangle_{sol} = \varnothing$ *for some* $C \in Pr$*.*

Proof. See Appendix B. ■

In the sequel, whenever we refer to the matching algorithm, we tacitly assume that Definition 5.3.4 is also included as part of the three-phase strategy, unless stated otherwise.

5.3.2 Correctness of the matching algorithm

In this section, soundness and completeness of the matching algorithm is proved.

The following lemma states that solutions to matching problems are preserved by α-equivalence. This Lemma is a variant of Lemma 2.5.13.

Lemma 5.3.8. *Given a matching problem Pr, assume* $\Delta \vdash X\sigma \approx_\alpha X\sigma'$ *for all* $X \in V(Pr)$*, for all* $\Delta \in \mathbf{F}$ *where* $\mathbf{F}$ *is a collection of freshness contexts. Then,* $(\mathbf{F},\sigma) \in \mathscr{U}(Pr)$ *if and only if* $(\mathbf{F},\sigma') \in \mathscr{U}(Pr)$*.*

Proof. It follows by definition of solution to a matching problem as given in Definition 5.1.1, and Lemma 2.5.13. The proof can be found in Appendix B. ■

Lemma 5.3.9 and Lemma 5.3.10 are technical results used in the proofs of Lemma 5.3.11 and Lemma 5.3.12 respectively. Lemma 5.3.9 shows the correspondence between function Ψ from Definition 5.2.1 and the disagreement set given in Equation 2.4, whereas Lemma 5.3.10 does the same between function DNF as given in the constraint checking algorithm and function `fresh` as given in Equation 2.2.

Lemma 5.3.9. *Given a set of matching problems* $\Psi(\phi\hat{}\pi, \phi'\hat{}\pi', \varnothing)^A_X$ *(see Definition 5.2.1), a set of atoms* $\mathtt{ds}(\nabla,(\phi\sigma)\hat{}\pi,(\phi'\sigma)\hat{}\pi')$ *(see Equation 2.4), some freshness context* ∇ *and v-substitution* σ *such that* σ *satisfies* ∇ *(see Property 4.1.7), then* $\nabla \vdash a\#\sigma(X)$ *for each* $a \in \mathtt{ds}(\nabla,(\phi\sigma)\hat{}\pi,(\phi'\sigma)\hat{}\pi') \iff \nabla \vdash Pr'\sigma$ *for some* $Pr \in \Psi(\phi\hat{}\pi, \phi'\hat{}\pi', \varnothing)^A_X$ *where* Pr' *is* Pr *with constraints of form* $s \stackrel{?}{\approx}_? t \in Pr$ *replaced by constraints of form* $s \approx_\alpha t$ *in* Pr'*.*

Proof. See Appendix B. ■

Lemma 5.3.10. *Given the set of formulæ* $\mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} \langle \pi^{-1}(b), a\#\phi(b), X\rangle))$ *(see Definition 3.2.9 for function* $\mathbf{S}_{nf}$ *and Definition 3.3.4 for function* $\langle\cdot,\cdot,\cdot\rangle$*), the set of atoms* $\mathtt{fresh}(\nabla,a,(\phi\sigma)\hat{}\pi)$ *(see Equation 2.2), some freshness context* ∇ *and v-substitution* σ *such that* σ *satisfies* ∇ *(see Property 4.1.7), then* $\nabla \vdash n\#\sigma(X)$ *for each* $n \in \mathtt{fresh}(\nabla,a,(\phi\sigma)\hat{}\pi)$ $\iff \nabla \vdash \Delta\sigma$ *for some* $\Delta \in \mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} \langle \pi^{-1}(b), a\#\phi(b), X\rangle))$*.*

Proof. See Appendix B. ■

Below, Lemma 5.3.11 and Lemma 5.3.12 state that solutions are preserved under non-instantiating and freshness rules respectively. Proofs were already provided for their standard counterparts in (Urban et al., 2004). They offer support for Theorem 5.3.17.

Lemma 5.3.11. *Let* Pr_0 *be a matching problem,* $Xs = V_{RHS}(Pr_0)$*. Then, consider a step* $(Pr,\theta) \stackrel{Xs}{\Longrightarrow}_{?\approx} \{(Pr'_i,\theta) | i \in I\}$ *in its reduction to normal form* $\langle Pr_0\rangle_{\mathtt{nf}_{?\approx}}$ *by application of some non-instantiating rule, then* $\mathscr{U}(Pr) = \bigcup_{i\in I} \mathscr{U}(Pr'_i)$*.*

Proof. By case analysis on the non-instantiating rules for the relation $\Longrightarrow_{?\approx}$. For the sake of simplicity, suppose $Pr = \{C\}$, that is, Pr contains only one problem. The interesting case is that of variables. The rest of the cases can be found in Appendix B.

- The case $(?\approx\mathbf{X})$. Suppose $(\{\phi\hat{}\pi\cdot X \;?\approx_?\; \phi'\hat{}\pi'\cdot X\},\theta) \overset{(?\approx\mathbf{X})}{\Longrightarrow} \bigcup_{Pr'_i\in\Psi(\phi\hat{}\pi,\phi\hat{}\pi,\varnothing)^A_X} \{(Pr'_i,\theta)\}$.

  Suppose also that $(\mathbf{F},\sigma_i)\in\mathscr{U}(Pr'_i)$ for each $Pr'_i\in\Psi(\phi\hat{}\pi,\phi\hat{}\pi,\varnothing)^A_X$. By Definition 4.1.8 we have, $\forall\nabla\in\mathbf{F}.\nabla\vdash Pr'^{\approx}_i\sigma_i$ where $Pr'^{\approx}_i$ is Pr'_i with constraints of form $s\;?\approx_?\;t$ replaced by constraints of form $s\approx_\alpha t$. Using Lemma 5.3.9 we obtain $\nabla\vdash a\#\sigma_i(X)$ for each $a\in\mathtt{ds}(\nabla,(\phi\sigma_i)\hat{}\pi,(\phi'\sigma_i)\hat{}\pi')$ such that, applying Lemma 2.5.14 we are able to obtain $\nabla\vdash((\pi\cdot\sigma_i(X))\phi\sigma_i)\approx_\alpha((\pi'\cdot\sigma_i(X))\phi'\sigma_i)$. Applying Definition 2.3.9 on both sides of $\approx_\alpha$ we obtain, $\nabla\vdash(\phi\hat{}\pi\cdot X)\sigma_i\approx_\alpha(\phi'\hat{}\pi'\cdot X)\sigma_i$. The result follows by Definition 4.1.8 and the union of sets.

  Otherwise, suppose $(\mathbf{F},\sigma)\in\mathscr{U}(\phi\hat{}\pi\cdot X\;?\approx_?\;\phi\hat{}\pi\cdot X)$. By Definition 4.1.8 we obtain, $\forall\nabla\in\mathbf{F}.\nabla\vdash(\phi\hat{}\pi\cdot X)\sigma\approx_\alpha(\phi'\hat{}\pi'\cdot X)\sigma$. By Definition 2.3.9 we have, $\nabla\vdash((\pi\cdot\sigma(X))\phi\sigma)\approx_\alpha((\pi'\cdot\sigma(X))\phi'\sigma)$. Using Lemma 2.5.15 we obtain, $\nabla\vdash a\#\sigma(X)$ for each $a\in\mathtt{ds}(\nabla,(\phi\sigma)\hat{}\pi,(\phi'\sigma)\hat{}\pi')$. Using Lemma 5.3.9 we have, $\nabla\vdash Pr'^{\approx}_i\sigma$ for each $Pr'_i\in\Psi(\phi\hat{}\pi,\phi\hat{}\pi,\varnothing)^A_X$ where $Pr'^{\approx}_i$ is Pr'_i with constraints of form $s\;?\approx_?\;t$ replaced by constraints of form $s\approx_\alpha t$. The result follows by Definition 4.1.8.

■

Lemma 5.3.12. *Assume* $(Pr_0,\theta)\Longrightarrow^*_{\#}(\langle Pr_0\rangle_{\mathtt{nf}},\theta)$ *where* Pr_0 *is a set of freshness constraints and consider a step* $(\{Pr_{i1},\ldots,Pr_{in}\},\theta)\Longrightarrow_{\#}(\{Pr'_{i1},\ldots,Pr'_{im}\},\theta)$ *by application of some freshness rule, then* $\bigcup_{k\in1\ldots n}\mathscr{U}(Pr_{ik})=\bigcup_{k'\in1\ldots m}\mathscr{U}(Pr'_{ik'})$.

Proof. By case analysis on the set of freshness rules for relation $\Longrightarrow_{\#}$. For the sake of simplicity, suppose $k=1$ such that $(\{Pr_{i1},\ldots,Pr_{in}\},\theta)=(\{Pr_{i1}\},\theta)$ and also $Pr_{i1}=\{C\}$, that is, Pr_{i1} contains only one problem. The interesting case is that of the variable. Other cases can be found in Appendix B.

- The case $(\#\mathbf{X})$. Suppose $Pr_{i1}=\{a\#\phi\hat{}\pi\cdot X\}$. Then, $(a\#\phi\hat{}\pi\cdot X,\theta)\overset{(\#\mathbf{X})}{\Longrightarrow}(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})}\langle\,\pi^{-1}(b),a\#\phi(b),X\,\rangle),\theta)$ where $\phi\neq\mathtt{Id}$ or $\pi\neq\mathtt{Id}$ by the side conditions imposed to the rule. By Definition 3.3.4 we observe that each $\langle\,\pi^{-1}(b),a\#\phi(b),X\,\rangle$ is a DNF of consistent freshness constraints, where $b\in(\mathscr{D}om(\phi)\cup\{a\})$. Then, following Lemma 3.3.8, it is the case that each $\Delta\in\mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})}\langle\,\pi^{-1}(b),a\#\phi(b),X\,\rangle))$ is a conjunctive clause of consistent freshness constraints.

  Suppose also that $(\mathbf{F}_i,\theta)\in\mathscr{U}(\Delta_i)$ for each $\Delta\in\mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})}\langle\,\pi^{-1}(b),a\#\phi(b),X\,\rangle))$. By Definition 4.1.8, $\forall\nabla_{ij}\in\mathbf{F}_i.\nabla_{ij}\vdash\Delta\theta$. Using Lemma 5.3.10 we have, $\nabla_{ij}\vdash n\#\theta(X)$ for each $n\in\mathtt{fresh}(\nabla,a,(\phi\theta)\hat{}\pi)$.

Applying Lemma 2.5.12 on the unpacked definition of $\mathtt{fresh}(\Delta_i, a, \phi\hat{}\pi)$ (see Equation 2.2). We are now able to apply Lemma 2.5.16 to obtain $\nabla_{ij} \vdash a\#(\phi\theta)\hat{}\pi\cdot\theta(X)$. By application of Definition 2.3.9 we have, $\nabla_{ij} \vdash a\#(\phi\hat{}\pi\cdot X)\theta$ and the result follows by Definition 4.1.8.

Otherwise, suppose $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#\phi\hat{}\pi\cdot X)$. By Definition 4.1.8 we have $\forall\nabla \in \mathbf{F}.\nabla \vdash a\#(\phi\hat{}\pi\cdot X)\theta$. Applying Definition 2.3.9 we have $\nabla \vdash a\#((\pi\cdot\theta(X)))\phi\theta$.
By Lemma 2.5.17, $\nabla \vdash n\#\theta(X)$ for each $n \in \mathtt{fresh}(\nabla, a, (\phi\theta)\hat{}\pi)$. Using Lemma 5.3.10 we have, $\nabla \vdash \Delta_i\theta$ for each $\Delta_i \in \mathbf{S}nf(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} \langle \pi^{-1}(b), a\#\phi(b), X \rangle))$. The result follows by Definition 4.1.8.

■

The technical lemma below supports correctness of the instantiating rules given in Lemma 5.3.15.

Lemma 5.3.13. *Given a matching problem Pr,* $(\mathbf{F}, \theta \bullet \sigma) \in \mathscr{U}(Pr)$ *if and only if* $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr\theta)$.

Proof. The result follows by the definition of solution to a matching problem (see Definition 5.1.1) and the definition of v-substitution composition where $\nabla \vdash P(\theta \bullet \sigma) \approx_\alpha (P\theta)\sigma$ for each $P \in Pr$ and each $\nabla \in \mathbf{F}$. ■

The next lemma also helps on providing support for Lemma 5.3.15 by showing that the matching algorithm gives correct results upon termination. This is the case because of function Cap which exhaustively provides a set of Cap terms to generate candidate solutions and the fact that variable terms on the RHS of $\stackrel{?}{\approx}$ are treated as constants with suspensions. Therefore, if a solution exists to a unification problem by application of an instantiating rule, the unifier contains one of the terms in the set produced by Cap. Additionally, when the constraint is of form $\phi\hat{}\pi\cdot X \stackrel{?}{\approx}_? \phi'\hat{}\pi'\cdot Y$, an additional solution involves renaming variable X to Y, taking into account suspended permutations.

Lemma 5.3.14. *Let* $\phi\hat{}\pi\cdot X$ *and* t *be a pair of terms such that* $V(\phi\hat{}\pi\cdot X) \cap V(t) = \varnothing$, σ *an idempotent v-substitution such that* $\sigma(X) \neq X$ *and* $\mathscr{D}om(\sigma) \cap V(t) = \varnothing$, *and let* ∇ *be a freshness context such that* σ *satisfies* ∇.

If $\nabla \vdash (\phi\hat{}\pi\cdot X)\sigma \approx_\alpha t$, *then* $\nabla \vdash u([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha u\sigma$ *where* u *is* $Z \in \mathscr{D}om(\sigma) \setminus \{X\}$ *or* $u = \phi\hat{}\pi\cdot X$ *and* s *is some term in* $\mathtt{Cap}(t, \mathscr{D}om(\phi))$ *whenever* $t \neq \phi'\hat{}\pi'\cdot Y$ *(resp. in* $\mathtt{Cap}(t, \mathscr{D}om(\phi)) \cup \{\pi'\cdot Y\}$ *whenever* $t = \phi'\hat{}\pi'\cdot Y$*).*

Proof. For any variable $Z \in \mathscr{D}om(\sigma)$ such that $Z \neq X$, the result follows trivially. The interesting case is when applying the v-substitution to $\phi\hat{}\pi\cdot X$. Then, we need to show that

(*a*) $\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha t$ for some $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$ whenever $t \neq \phi'\hat{}\pi'{\cdot}Y$;

(*b*) $\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$ for some $s \in \mathtt{Cap}(\phi'\hat{}\pi'{\cdot}Y, \mathscr{D}om(\phi))$ or $\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma) \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$.

Using Definition 2.3.9 we have,

(*a*) $\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha s(\phi([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma))$;

(*b*) $\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha s(\phi([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma))$ and $\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma) \approx_\alpha \phi([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma)\hat{}\pi'{\cdot}Y$.

Hence, by application of the Transitive property from Theorem 2.5.8, we are left to prove that

(*a*) $\nabla \vdash s(\phi([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma)) \approx_\alpha t$ for some $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$;

(*b*) $\nabla \vdash s(\phi([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma)) \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$ for some $s \in \mathtt{Cap}(\phi'\hat{}\pi'{\cdot}Y, \mathscr{D}om(\phi))$ or $\nabla \vdash \phi([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma)\hat{}\pi'{\cdot}Y \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$.

We show the work for the proof of part (a) since part (b) is similarly solved. However, the full proof can be found in Appendix B.

- For part (a).

  We distinguish two cases to prove, the case where $\phi = \mathtt{Id}$ and the case where $\phi \neq \mathtt{Id}$.

  • For the case where $\phi = \mathtt{Id}$.

  By Definition 5.2.3 we have, $\mathtt{Cap}(t, \varnothing) = \{t\}$ such that $s = t$. By Definition 2.3.9 we have, $\nabla \vdash t(\phi([X \mapsto \pi^{-1}{\cdot}t] \bullet \sigma)) \approx_\alpha t$. By the Reflexive property, $\nabla \vdash t \approx_\alpha t$ always. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha t$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi{\cdot}X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

  • For the case where $\phi \neq \mathtt{Id}$.

  For the sake of clarity, assume that $\phi(a) \neq a$ for atom a and $\phi(b) = b$ for $b \in (\mathscr{A} \setminus \{a\})$ (i.e., ϕ is a singleton) and also that $t|_k = a$ is the only occurrence of atom a in t, for some position $k \in \mathscr{P}os(t)$. More general cases are built by induction.

  We distinguish two cases, the case where $s = t$ and the case where $s \in \{t[a \cdots a]_{p_1 \cdots p_n} \mid a \in \mathscr{D}om(\phi), p_i \in \mathscr{P}os(t), 1 \leq i \leq n\}$. We assume without loss of generality that $i = 1$, i.e., $s = t[a]_p$ for some $t[a]_p \in (\{t[a]_p \mid a \in \mathscr{D}om(\phi), p \in \mathscr{P}os(t)\})$. Other cases where $i > 1$ are built by induction.

  1. Suppose $s = t$.

     Then, either $\nabla \vdash a\#t$ or $a \in supp(t)$ (see Definition 1.2.14), that is, a occurs unabstracted in t.

(a) Suppose that $\nabla \vdash a\#t$.
Then, by Definition 2.3.2 we obtain, $\nabla \vdash t(\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma)) \approx_\alpha t$ and $\nabla \vdash t \approx_\alpha t$ always. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha t$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

(b) Otherwise, if a occurs in t unabstracted at some position k, then $|t| < |t(\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma))|$ whenever $\nabla \vdash t|_k \phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma) \not\approx_\alpha t|_k$. Hence, it can only be that $\nabla \vdash t|_k \phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma) \approx_\alpha t|_k$ such that $\nabla \vdash t(\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma)) \approx_\alpha t$. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha t$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

2. Otherwise, suppose $s = t[a]_p$ for some $t[a]_p \in \{t[a]_p \mid a \in \mathscr{D}om(\phi), p \in \mathscr{P}os(t)\}$. Suppose also $t|_p = v$ and $\exists p' \in \mathscr{P}os(t)$ (e.g.: $p' = \varepsilon$) such that $t|_{p'} = w$ and $p' \leq p$ and $v \lhd w$, that is, v occurs in term w. Then, the judgement we need to prove is now represented as $\nabla \vdash (t[a]_p)(\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)) \approx_\alpha t[v]_p$.
There are two cases to show, the case where $\nabla \vdash a\#t[v]_p$ and the case where $a \in supp(t[v])_p$ (see Definition 1.2.14), that is, a occurs unabstracted in t.

(a) Suppose $\nabla \vdash a\#t[v]_p$.
Now, we have two more cases to show, either $(w = [a]l \wedge w \neq v)$ or $(w \neq [a]l \vee w = v)$.

i. Suppose that $w = [a]l \wedge w \neq v$ (hence $v \lhd l$).
Then, by Definition 2.3.2 we have, $\nabla \vdash (t[a]_p)(\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)) \approx_\alpha t[a]_p$. Hence, it can only be that $\nabla \vdash a \approx_\alpha v$ such that, by Corollary 2.5.20, $\nabla \vdash t[a]_p \approx_\alpha t[v]_p$. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[a]_p$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

ii. Suppose that $(w \neq [a]l \vee w = v)$.
Then, by Definition 2.3.2 we have $\nabla \vdash t[a]_p(\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p$. Then, it can only be that, at position p, $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha v$ such that, by Corollary 2.5.20, $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha t[v]_p$. Then, by the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$

where $t \equiv t[v]_p$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by another application of the Transitive property from Theorem 2.5.8.

(b) Otherwise, suppose $a \in supp(t)$ (see Definition 1.2.14), that is, a occurs unabstracted in t at position k (by the assumption).
We show two cases with respect to the structure of w, either $(w = [a]l \wedge w \neq v)$ or $(w \neq [a]l \vee w = v)$.

i. Suppose $w = [a]l \wedge w \neq v$ (hence $v \lhd l$).
By the assumption we observe there is only one occurrence of atom a in t occurring at position k and thus $t|_k \not\lhd w$ since $t|_k$ occurs unabstracted in $t\sigma$. Then, it must be the case that $k \parallel p$ so that we must prove $\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha (t[v]_p)[a]_k$. Then, by Definition 2.3.2 we obtain, $\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha ((t[a]_p)[\pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_k)$. Then, notice that $|(t|_k\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma))| = |(t|_k)|$ only for the case where
$\nabla \vdash t|_k\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t|_k$. Further, since v is in the scope of $[a]$- by the assumption, it can only be that $\nabla \vdash a \approx_\alpha v$ at position p. Therefore, by the Transitive property from Theorem 2.5.8 it follows that $\nabla \vdash \pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha a \approx_\alpha v$ and applying Corollary 2.5.20 we have, $\nabla \vdash ((t[a]_p)[\pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_k) \approx_\alpha (t[v]_p)[a]_k$. Then, by the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ where $t \equiv t[v]_p$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash ((t[a]_p)[\pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_k) \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

ii. Otherwise, suppose that $(w \neq [a]l \vee w = v)$.
Then, either $t|_k \lhd v$ or $t|_k \not\lhd v$ where $t|_k = a$ by the assumption.
• Suppose that $t|_k \lhd v$.
Then, one has to prove $\nabla \vdash (t[a]_p)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[v]_p$. By Definition 2.3.2 we have, $\nabla \vdash (t[a]_p)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p)$. Then, it must be the case that $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha v$ at position p such that, by Corollary 2.5.20, we obtain $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha t[v]_p$. Then, by the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ where $t \equiv t[v]_p$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$.

The result follows by another application of the Transitive property from Theorem 2.5.8.
• Otherwise, suppose $t|_k \not\lhd v$.
Then, one has to prove $\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha (t[v]_p)[a]_k$. By Definition 2.3.2 we have,
$\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha$
$(t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_p)[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_k)$.
Then, note that, at position p, $|(\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma))| \leq |v|$ only when $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha v$ and at position k, $|(\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma))| \leq |a|$ only when $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha a$. Therefore it must be the case that, by the Transitive property from Theorem 2.5.8, $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha v \approx_\alpha a$ such that, by Corollary 2.5.20, we obtain $\nabla \vdash (t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_p)[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_k) \approx_\alpha (t[v]_p)[a]_k$. Then, by the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ where $t \equiv (t[v]_p)[a]_k$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_p)[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_k) \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by another application of the Transitive property from Theorem 2.5.8.

Hence the property holds. ■

The following lemma establishes the correctness of the instantiating rules and supports Theorem 5.3.17.

Lemma 5.3.15. *Assume* $(Pr_0, \mathtt{Id}) \stackrel{Xs}{\Longrightarrow}_* \langle Pr_0 \rangle_{\mathtt{nf}_{?\approx}}$, *where* $XS = V_{RHS}(Pr_0)$, *and consider a step* $(Pr, \theta) \stackrel{Xs}{\Longrightarrow} \bigcup_{i \in I} \{(Pr'_i, \theta \bullet \theta'_i)\}$ *via some instantiating rule,*

1. *if* $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr)$, *then* $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr'_i)$ *and* $\forall \nabla \in \mathbf{F}.\nabla \vdash v(\theta'_i \bullet \sigma) \approx_\alpha v\sigma$ *for each constraint of form* $v \;?\approx_?\; w$ *in* Pr *and some* $i \in I$.

2. *for any* $i \in I$, *if* $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr'_i)$, *then* $(\mathbf{F}, \theta'_i \bullet \sigma) \in \mathscr{U}(Pr)$.

Proof. There are two instantiating rules to examine, rule ($?\approx$**Inst**) and rule ($?\approx$**XY**). We show the proof of rule ($?\approx$**Inst**). The other case is similar and can be found in Appendix B.

For rule ($?\approx$**Inst**) there is the following instantiation step:
Let $Pr = \{\phi\hat{}\pi\cdot X \;?\approx_?\; t\} \cup Pr''$ where $t \neq \phi'\hat{}\pi'\cdot Y$ for some $Y \in \mathscr{X}$ and θ a v-substitution. Then, $(Pr, \theta) \stackrel{X \notin Xs}{\Longrightarrow}_{(?\approx\textbf{Inst})}$
$\bigcup_{s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))} (\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \;?\approx_?\; t\} \cup Pr''[X \mapsto \pi^{-1}\cdot s])$ where $X \notin V(s)$ by definition of

matching solution (see Definition 5.1.1) and function Cap (see Definition 5.2.3) for each $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$.

For the first part:

($_?\approx$**Inst**) Suppose that $(\mathbf{F}, \sigma) \in \mathscr{U}(\phi\hat{}\pi{\cdot}X \;{}_?{\approx}_?\; t, Pr'')$. By Definition 4.1.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi{\cdot}X)\sigma \approx_\alpha t\sigma, Pr''\sigma$ where, by Definition 5.1.1, it is the case that $V(\phi\hat{}\pi{\cdot}X) \cap V(t) = \varnothing$ and $\mathscr{D}om(\sigma) \cap V(t) = \varnothing$ so that, by Lemma 5.3.14 we obtain, $\forall \nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha (\phi\hat{}\pi{\cdot}X)\sigma$ and $\forall \nabla \in \mathbf{F}.\nabla \vdash ([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha \sigma$ for all $Y \in \mathscr{D}om(\sigma) \setminus \{X\}$ and some term s such that $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$. By composition of v-substitutions we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi{\cdot}X)([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha ((\phi\hat{}\pi{\cdot}X)[X \mapsto \pi^{-1}{\cdot}s])\sigma$. By Definition 2.3.9 we have,
$\forall \nabla \in \mathbf{F}.\nabla \vdash ((\phi\hat{}\pi{\cdot}X)[X \mapsto \pi^{-1}{\cdot}s])\sigma \approx_\alpha (s(\phi[X \mapsto \pi^{-1}{\cdot}s]))\sigma$. As a result of $\mathscr{D}om(\sigma) \subseteq V(Pr)$ and applying Lemma 2.5.13 we obtain, $\forall \nabla \in \mathbf{F}.\nabla \vdash Pr''\sigma \approx_\alpha Pr''([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma)$. By composition of v-substitutions we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash Pr''([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma) \approx_\alpha (Pr''([X \mapsto \pi^{-1}{\cdot}s])\sigma$. Using the Transitive property from Theorem 2.5.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash (s(\phi[X \mapsto \pi^{-1}{\cdot}s]))\sigma \approx_\alpha t\sigma, (Pr''([X \mapsto \pi^{-1}{\cdot}s])\sigma)$. The result follows by Definition 4.1.8.

For the second part,

($_?\approx$**Inst**) Suppose that, for any $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$, $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr'_s)$ where $Pr'_s = s(\phi\theta') \;{}_?{\approx}_?\; t, Pr''\theta'$ and $\theta' = [X \mapsto \pi^{-1}{\cdot}s]$. By application of Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash (s(\phi\theta'))\sigma \approx_\alpha t\sigma, (Pr''\theta')\sigma$ where by Definition 5.1.1, $\mathscr{D}om(\sigma) \cap V(t) = \varnothing$. Thus by Lemma 2.5.13 we have, trivially, $\forall \nabla \in \mathbf{F}.\nabla \vdash t\sigma \approx_\alpha t(\theta' \bullet \sigma)$. Applying composition of v-substitutions, $\forall \nabla \in \mathbf{F}.\nabla \vdash t(\theta' \bullet \sigma) \approx_\alpha (t\theta')\sigma$ and by the Transitive property from Theorem 2.5.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash (s(\phi\theta'))\sigma \approx_\alpha (t\theta')\sigma, (Pr''\theta')\sigma$. Now, by claim 2. of Lemma 2.5.4 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash (s(\phi\theta'))\sigma \approx_\alpha (\pi \circ \pi^{-1}){\cdot}((s(\phi\theta'))\sigma)$. Applying Property 2.5.10 we obtain, $\forall \nabla \in \mathbf{F}.\nabla \vdash (\pi \circ \pi^{-1}){\cdot}((s(\phi\theta'))\sigma) \approx_\alpha ((\pi \circ \pi^{-1}){\cdot}(s(\phi\theta')))\sigma$.
Applying Property 2.5.9 we have,
$\forall \nabla \in \mathbf{F}.\nabla \vdash ((\pi \circ \pi^{-1}){\cdot}(s(\phi\theta')))\sigma \approx_\alpha (((\pi \circ \pi^{-1}){\cdot}s)((\pi \circ \pi^{-1}){\cdot}(\phi\theta')))\sigma$.
Since $(\pi \circ \pi^{-1}) = \mathtt{Id}$ we observe that,
$\forall \nabla \in \mathbf{F}.\mathtt{ds}(\nabla, ((\pi \circ \pi^{-1}){\cdot}(\phi\theta'))\hat{}(\pi \circ \pi^{-1}), (\phi\theta')\hat{}(\pi \circ \pi^{-1})) = \varnothing$ such that, by
Lemma 2.5.14 we obtain,
$\forall \nabla \in \mathbf{F}.\nabla \vdash (((\pi \circ \pi^{-1}){\cdot}s)((\pi \circ \pi^{-1}){\cdot}(\phi\theta')))\sigma \approx_\alpha (((\pi \circ \pi^{-1}){\cdot}s)(\phi\theta'))\sigma$. By Definition 2.1.8, $(((\pi \circ \pi^{-1}){\cdot}s)(\phi\theta'))\sigma = ((\pi{\cdot}(\pi^{-1}{\cdot}s))(\phi\theta'))\sigma$. By the assumption, $((\pi{\cdot}(\pi^{-1}{\cdot}s))(\phi\theta'))\sigma = ((\pi{\cdot}\theta'(X))(\phi\theta'))\sigma$. Using Definition 2.3.9, $\forall \nabla \in \mathbf{F}.\nabla \vdash ((\pi{\cdot}\theta'(X))(\phi\theta'))\sigma \approx_\alpha ((\phi\hat{}\pi{\cdot}X)\theta')\sigma$. Applying the Transitive property from Theorem 2.5.8 we obtain, $\forall \nabla \in \mathbf{F}.\nabla \vdash ((\phi\hat{}\pi{\cdot}X)\theta')\sigma \approx_\alpha (t\theta')\sigma, (Pr''\theta')\sigma$. By Defini-

tion 4.1.8 we obtain, $(\mathbf{F}', \sigma) \in \mathscr{U}((\phi\hat{}\pi \cdot X)\theta' \approx_\alpha t\theta', Pr''\theta')$. The result follows by Lemma 5.3.13.

■

The following technical lemma will be applied in the proof of Theorem 5.3.17 below. It states that any v-substitution instantiating a variable $X \in \mathscr{X}$ can be refactored into a v-substitution which is only non-trivial on X and a v-substitution that is trivial on X.

Lemma 5.3.16. *Suppose $\Delta \vdash X\sigma \approx_\alpha s\sigma$ and $X \notin V(s)$ for some variable X, v-substitution σ, freshness context Δ and term s. Then,*

- $\Delta \vdash X\sigma \approx_\alpha X([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$*, and also*
- $\Delta \vdash Y\sigma \approx_\alpha Y([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$ *for $Y \in \mathscr{X} \setminus \{X\}$.*

Proof. See Appendix B. ■

The following theorem demonstrates that the unification algorithm does indeed produce unifiers to any unifiable problem Pr and that the unifiers are idempotent. Further, the set of solutions calculated by the unification algorithm is such that any other solution to Pr is an instance of some pair occurring in the unification result. The proof is now easy to derive by application of Lemma 5.3.15 establishing the correctness of the instantiating rules.

Theorem 5.3.17 (Soundness and completeness)**.** *Given a unifiable matching problem Pr and $\langle Pr\rangle_{sol} = \{(\mathbf{F}_i, \theta_i) \mid i \in I\}$ where $(\mathbf{F}_i, \theta_i)$ consists of a collection of freshness contexts $\mathbf{F}_i$ and v-substitution θ_i for $i \in I$,*

1. *$\langle Pr\rangle_{sol} \subseteq \mathscr{U}(Pr)$(soundness); and*
2. *$\forall(\mathbf{F}, \sigma) \in \mathscr{U}(Pr)$, $\exists(\mathbf{F}_i, \theta_i) \in \langle Pr\rangle_{sol}$ such that $(\mathbf{F}_i, \theta_i) \leq (\mathbf{F}, \sigma)$ (completeness).*

Proof. By induction on the length of the reduction sequence $\{(Pr, \mathtt{Id})\} \stackrel{Xs}{\Longrightarrow}_* \langle Pr\rangle_{sol}$ where $Xs = V_{RHS}(Pr)$.

Both parts are proved together, by induction on the length of the reduction sequence. There are two main cases depending on the length of the reduction, the case for length 0 and the case for length $n+1$. For the latter, we also distinguish between phase 1 and phase 2 of the reduction sequence as stated in Definition 5.4.6. The application of function $[\cdot]$ to the normal form resulting from the reduction sequence has been omitted since it is not relevant to prove the matching process is both sound and complete.

- Length 0.

  Then, either $Pr = \varnothing$ or Pr is a set of primitive constraints, ∇.

– If $Pr = \varnothing$, then $\{(\varnothing, \mathtt{Id})\} = \langle\varnothing\rangle_{sol}$.

1. Trivially, $\vdash \varnothing\mathtt{Id}$ and also $\mathtt{Id}$ is idempotent. The result follows by Definition 4.1.8.
2. Suppose also that $(\mathbf{F}, \sigma) \in \mathscr{U}(\varnothing)$. Trivially, we observe that for all $\Delta \in \mathbf{F}$ $\Delta \vdash \mathtt{Id} \bullet \sigma \approx_\alpha \sigma$ and $\Delta \vdash \varnothing\sigma$ such that, by Definition 4.1.14 we obtain, $(\{\varnothing\}, \mathtt{Id}) \leq (\mathbf{F}, \sigma)$. The result follows.

– If $Pr = \nabla$, then by Lemma 5.3.7 we have, $\{(\nabla, \mathtt{Id})\} = \langle\nabla\rangle_{sol}$.

1. Trivially, $\nabla \vdash \nabla\mathtt{Id}$. The result follows by Definition 4.1.8.
2. Suppose also that $(\mathbf{F}, \sigma) \in \mathscr{U}(\nabla)$. Trivially, we observe that, for all $\Delta \in \mathbf{F}$, $\Delta \vdash \mathtt{Id} \bullet \sigma \approx_\alpha \sigma$ and $\Delta \vdash \nabla\sigma$ by Definition 4.1.8, such that, by Definition 4.1.14 we obtain, $(\nabla, \mathtt{Id}) \leq (\mathbf{F}, \sigma)$. The result follows.

• Length $n+1$.

We examine first the case for phase 2 and then continue by constructing a proof for phase 1 of the reduction sequence. During phase 1 we have five cases to examine, the non-instantiating case and the four instantiating cases; during phase 2 there is only a single case to examine.

Phase 2.

– Suppose that Pr is a set of freshness constraints. Suppose also that $(Pr, \mathtt{Id}) \overset{\Omega}{\Longrightarrow} (\{Pr_1, \ldots, Pr_n\}, \mathtt{Id}) \Longrightarrow^*_{\#} \bigcup_{k \in 1 \ldots n} \langle Pr_k\rangle_{sol}$ by some freshness rule Ω.

1. By inductive hypothesis, $\langle Pr_k\rangle_{sol} \subseteq \mathscr{U}(Pr_k), (k \in 1 \ldots n)$. By Lemma 5.3.12, $\mathscr{U}(Pr) = \bigcup_{k \in 1 \ldots n} \mathscr{U}(Pr_k)$ and the result follows.
2. By inductive hypothesis, $\forall(\mathbf{F}, \sigma) \in \mathscr{U}(Pr_k), \exists(\mathbf{F}_i, \mathtt{Id}) \in \langle Pr_k\rangle_{sol} (k \in 1 \ldots n)$ such that $(\mathbf{F}_i, \mathtt{Id}) \leq (\mathbf{F}, \sigma)$. By Lemma 5.3.12, $\mathscr{U}(Pr) = \bigcup_{k \in 1 \ldots n} \mathscr{U}(Pr_k)$ and the result follows.

Phase 1.

– Suppose $(Pr, \theta) \overset{\Omega}{\Longrightarrow} \{(Pr_k, \theta') \mid K \in \mathrm{I}\} \Longrightarrow^* \bigcup_{k \in \mathrm{I}} \langle Pr_k\rangle_{sol}$ by some non-instantiating rule Ω.

1. By inductive hypothesis, $\langle Pr_k\rangle_{sol} \subseteq \mathscr{U}(Pr_k)$(for $k \in \mathrm{I}$). By Lemma 5.3.11, $\langle Pr\rangle_{sol} = \bigcup_{k \in \mathrm{I}} \langle Pr_k\rangle_{sol}$. The result then follows.

2. By inductive hypothesis, $\forall(\mathbf{F},\sigma) \in \mathscr{U}(Pr_k)$, $\exists(\mathbf{F}_i,\theta_i) \in \langle Pr_k\rangle_{sol}$(for $k \in \mathrm{I}$) such that $(\mathbf{F}_i,\theta_i) \leq (\mathbf{F},\sigma)$. By Lemma 5.3.11, $\langle Pr\rangle_{sol} = \bigcup_{k\in\mathrm{I}} \langle Pr_k\rangle_{sol}$ and the result follows.

– Suppose $Pr = \{\phi\hat{}\pi\cdot X \;_?{\approx}_?\; t\} \cup Pr''$ where $t \neq \phi'\hat{}\pi'\cdot Y$ for any $Y \in \mathscr{X}$ such that $(Pr,\theta) \overset{X\notin Xs}{\Longrightarrow}_{(?\approx\textbf{Inst})}$

$$\bigcup_{s\in\mathtt{Cap}(t,\mathscr{D}om(\phi))} \{(\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s], \theta\bullet[X\mapsto\pi^{-1}\cdot s])\}$$

$$\overset{Xs}{\Longrightarrow}_*$$

$$\bigcup_{s\in\mathtt{Cap}(t,\mathscr{D}om(\phi))} \langle(\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s], \theta\bullet[X\mapsto\pi^{-1}\cdot s])\rangle_{sol}$$

1. By inductive hypothesis, for each $s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$, $\langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol} \subseteq \mathscr{U}(\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s])$ where each v-substitution $[X\mapsto\pi^{-1}\cdot s]$ is idempotent by Definition of matching problem (see Definition 5.1.1) and definition of function Cap (see Definition 5.2.3)..

   Now, by Lemma 5.3.7 we observe there are two cases to examine: either

   (a) $\langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol} = \varnothing$ for some (possibly none) $s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$ or

   (b) $\langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol} = \{((\mathbf{F}_i,\theta_i) \mid i \in \mathrm{I})\}$ for some term $s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$ where each pair $(\mathbf{F}_i,\sigma_i)$ consists of a collection of freshness contexts $\mathbf{F}_i$ and an idempotent unifier θ_i. Further, by the assumptions it is the case that the given matching problem is unifiable and therefore $\forall s.s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$, $\exists\langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol}$ such that $\langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol} \neq \varnothing$.

   Then, for case (a) we have, trivially, $\varnothing \subset \mathscr{U}(Pr)$ and for case (b), by claim 2 of Lemma 5.3.15 we obtain, $\{(\mathbf{F}_i,([X\mapsto\pi^{-1}\cdot s]\bullet\theta_i)) \mid i \in \mathrm{I}\} \subseteq \mathscr{U}(Pr)$ for all $\langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol} = \{((\mathbf{F}_i,\theta_i) \mid i \in \mathrm{I})\}$ where $s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$ and, it is easy to see that, $([X\mapsto\pi^{-1}\cdot s]\bullet\theta_i)$ is idempotent. Hence it is the case that $\langle Pr\rangle_{sol} = \bigcup_{s\in\mathtt{Cap}(t,\mathscr{D}om(\phi))} \langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol}$ and the result then follows.

2. Suppose $(\mathbf{F},\sigma) \in \mathscr{U}(Pr)$. By claim 1 of Lemma 5.3.15 we obtain, $(\mathbf{F},\sigma) \in \mathscr{U}(\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s])$ and $\forall\Delta \in \mathbf{F}.\Delta \vdash v([X\mapsto\pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha v\sigma, w([X\mapsto\pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha w\sigma$ for each $v \;_?{\approx}_?\; w \in Pr$ and some term $s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$ such that $\langle\{s(\phi[X\mapsto\pi^{-1}\cdot s]) \;_?{\approx}_?\; t\} \cup Pr''[X\mapsto\pi^{-1}\cdot s]\rangle_{sol} \neq \varnothing$ (the assumption ensures there is at least one). For

any other case (possibly none) such that $\langle\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \,?\approx_?\, t\} \cup Pr''[X \mapsto \pi^{-1}\cdot s]\rangle_{sol} = \varnothing$, there is nothing to prove. Then, by Lemma 5.3.16 we have $\forall \Delta \in \mathbf{F}.\Delta \vdash [X \mapsto \pi^{-1}\cdot s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}} \approx_\alpha \sigma$. By Lemma 5.3.8, $(\mathbf{F}, [X \mapsto \pi^{-1}\cdot s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}}) \in \mathscr{U}(Pr)$. By application of Lemma 5.3.13 we obtain, $(\mathbf{F}, \sigma_{\mathscr{D}om(\sigma)\setminus\{X\}}) \in \mathscr{U}(\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \,?\approx_?\, t\} \cup Pr''[X \mapsto \pi^{-1}\cdot s])$. Now, by the inductive hypothesis we have, $\exists(\mathbf{F}_i, \theta_i) \in \langle\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \,?\approx_?\, t[X \mapsto \pi^{-1}\cdot s]\} \cup Pr''[X \mapsto \pi^{-1}\cdot s]\rangle_{sol}$ such that $(\mathbf{F}_i, \theta_i) \leq (\mathbf{F}, \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$. By the soundness claim, and following the notation given in its proof, it is a fact that $(\mathbf{F}_i, ([X \mapsto \pi^{-1}\cdot s] \bullet \theta_i)) \in \langle Pr\rangle_{sol}$ and $\langle Pr\rangle_{sol} \subseteq \mathscr{U}(Pr)$. By application of Lemma 4.1.17 we obtain, $(\mathbf{F}_i, [X \mapsto \pi^{-1}\cdot s] \bullet \theta_i) \leq (\mathbf{F}, [X \mapsto \pi^{-1}\cdot s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$. Using Lemma 5.3.8 we have, $(\mathbf{F}_i, [X \mapsto \pi^{-1}\cdot s] \bullet \theta_i) \leq (\mathbf{F}, [X \mapsto \pi^{-1}\cdot s] \bullet \sigma)$. The result follows.

– Suppose $Pr = \{\phi\hat{}\pi\cdot X \,?\approx_?\, \phi'\hat{}\pi'\cdot Y\} \cup Pr''$ where $X \neq Y$ such that
$(Pr, \theta) \overset{X\notin Xs}{\Longrightarrow}_{(?\approx \mathbf{XY})}$
$$\bigcup_{s\in(\mathtt{Cap}(\phi'\hat{}\pi'\cdot Y, \mathscr{D}om(\phi))\cup\{\pi'\cdot Y\})} (\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \,?\approx_?\, \phi'\hat{}\pi'\cdot Y\} \cup Pr''[X \mapsto \pi^{-1}\cdot s], \theta \bullet [X \mapsto \pi^{-1}\cdot s])$$
$\overset{Xs}{\Longrightarrow}_*$
$$\bigcup_{s\in(\mathtt{Cap}(\phi'\hat{}\pi'\cdot Y, \mathscr{D}om(\phi))\cup\{\pi'\cdot Y\})} \langle\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \,?\approx_?\, \phi'\hat{}\pi'\cdot Y\} \cup Pr''[X \mapsto \pi^{-1}\cdot s]\rangle_{sol}.$$
The result follows similarly to case $Pr = \{\phi\hat{}\pi\cdot X \,?\approx_?\, t\} \cup Pr''$ for both claims and thus omitted.

Hence, the result follows. ■

Theorem 5.3.18 shows that the matching algorithm is *correct*, that is, if a solution to a matching problem exists, then the matching algorithm will find the complete set of principal solutions to such matching problem. However, if a matching problem is unsolvable, then the matching algorithm will return the empty set.

Theorem 5.3.18 (Correctness of the matching algorithm)**.** *Given a matching problem Pr, if the matching algorithm from Definition 5.2.6 succeeds then the process returns a complete set of principal solutions, otherwise the problem is unsolvable and the algorithm returns* $\varnothing$.

Proof. In the case that the matching algorithm succeeds, we apply Theorem 5.3.17. Otherwise, during every distinct sequence of transformations (possibly one) to matching problem *Pr*, the matching algorithm generates inconsistent constraints of form $a\#a$ or clashing equalities of form $f(s_1,\ldots,s_n) \,?\approx_?\, f(t_1,\ldots,t_m)$ where $n \neq m$, $f(s_1,\ldots,s_n) \,?\approx_?\, g(t_1,\ldots,t_m)$, $f(s_1,\ldots,s_n) \,?\approx_?\, [a]s$, $f(s_1,\ldots,s_n) \,?\approx_?\, a$, $[a]s \,?\approx_?\, b$, $a \,?\approx_?\, b$ or $\phi\hat{}\pi\cdot X \,?\approx_?\, t$ where $X \in$

$V_{RHS}(Pr)$ and $t \neq \phi'^{\frown}\pi' \cdot X$. It is a fact that none of these have a solution and by application of rule $(?\approx\bot)$ to clashing equalities and $(\#\bot)$ to inconsistent constraints, the set of solutions is reduced to empty. ■

In order to induce rewriting steps on the extended formalism, we are interested on finding a class of extended terms for which the matching algorithm finds, at most, a most general unifier to a pattern matching problem. This is done in the next section.

5.4 A Unitary Matching Algorithm

The chapter continues by applying further constraints to both the matching algorithm given in Definition 5.2.6 and the set of matching constraints in order to find a class of problems for which matching finds at most one principal solution. This *simple-matching algorithm* will be applied in the next chapter when providing a means of mechanising extended nominal rewriting.

In the sequel, we referred to the matching algorithm given in Definition 5.2.6 as *general matching algorithm*, to distinguish it from the simple-matching algorithm defined later in Definition 5.4.6.

5.4.1 Simple matching problems

The presence of atom substitutions suspended over variables may lead to a number of distinct most general unifiers for a given matching problem. Consider the following matching constraint

$$[a \mapsto c] \cdot X \ ?\approx c$$

A possible solution is $(\varnothing, [X \mapsto c])$. Another solution is $(\varnothing, [X \mapsto a])$. Although both solutions are correct and principal ones, when solving matching problems to generate rewrite steps we aim to have at most one principal solution. To recover the property of unique most general unifier we follow (Fairweather et al., 2015) where constraints with non-trivial a-substitutions are postponed as equality constraints. Then, solutions to the unification problem are regarded as conditional with respect to these constraints. Conditional solutions are thus derived from decidable formulæ occurring during the matching process.

Definition 5.4.1 and Remark 5.4.3 adapt the work in (Fairweather et al., 2015).

Definition 5.4.1 (Simple matching problem)**.** A **simple (nominal) term** s is a term where $V(s) \subseteq V_f(s)$, that is, in a simple term each variable symbol $X \in V(s)$ has an occurrence of the form $\pi \cdot X$ that is not in a suspended a-substitution. Then, call **simple matching constraint**

the equation of form $s \stackrel{?}{\approx} t$ where s is a simple term. A simple matching constraint is denoted by $s \stackrel{s}{?}\approx t$. Additionally, call **simple** the matching problem Pr if $(\ldots, s_i, \ldots) \stackrel{s}{?}\approx (\ldots, t_i, \ldots)$ for each matching constraint $s_i \stackrel{?}{\approx} t_i$ in Pr.

A solution to a simple matching problem is also a solution to a matching problem as in Definition 5.1.1. The change in representation (i.e., $s \stackrel{s}{?}\approx t$ if $s \stackrel{?}{\approx} t$ is simple) is useful to denote that solutions to simple matching problems are obtained by application of a more restrictive matching algorithm (i.e., a simple-matching algorithm to be introduced in Definition 5.4.6). We remark on this again after the definition of the simple-matching algorithm.

Example 5.4.2.

- $\mathtt{fn}([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}, \mathtt{X}, (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}) \stackrel{s}{?}\approx \mathtt{fn}([\mathtt{a}]\mathtt{a}, [\mathtt{b}]\mathtt{Z}, \mathtt{g}(\mathtt{b}))$ is a simple matching constraint but neither is $\mathtt{fn}([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}, \mathtt{X}) \stackrel{?}{\approx} \mathtt{fn}([\mathtt{a}]\mathtt{a}, [\mathtt{b}]\mathtt{Z})$ nor $\mathtt{fn}([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}, (\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}) \stackrel{?}{\approx} \mathtt{fn}([\mathtt{a}]\mathtt{a}, \mathtt{g}(\mathtt{b}))$. The former does not have a fixed occurrence of variable $\mathtt{Y}$ whereas the latter has a fixed occurrence of variable $\mathtt{X}$ but the suspended a-substitution is not trivial.

- $\{([\mathtt{a}]\mathtt{Y}, [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Y}) \stackrel{s}{?}\approx ([\mathtt{b}]\mathtt{b}, [\mathtt{a}]\mathtt{X})\}$ is a simple matching problem. The matching problem $\{\mathtt{f}([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}) \stackrel{s}{?}\approx \mathtt{f}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Y}), \mathtt{g}(\mathtt{X}) \stackrel{s}{?}\approx \mathtt{g}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{Y})\}$ is also simple since tuple $\{(\mathtt{f}([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}), \mathtt{g}(\mathtt{X})) \stackrel{s}{?}\approx (\mathtt{f}((\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}), \mathtt{g}((\mathtt{a}\ \mathtt{b})\cdot\mathtt{Y}))\}$ is a simple matching constraint.

For the set of simple matching constraints there is at most one principal solution. We prove this claim in Theorem 5.4.11. Next, a distinction must be made in the theory between equations solvable by the set of simple matching rules to be introduced in Definition 5.4.6, equations postponed until a v-substitution is available and clashing equalities (see Definition 5.2.5). To that extent, the definition below characterises postponed matching constraints.

Remark 5.4.3 (Postponed constraints). Given a simple matching problem Pr, matching constraints of form $\phi\hat{}\pi\cdot X \stackrel{s}{?}\approx t \in Pr$ where $t \neq \phi'\hat{}\pi'\cdot X$, $\phi \neq \mathtt{Id}$ and $X \notin V_{RHS}(Pr)$ are delayed until an instantiation for X is readily available; Definition 5.4.1 ensures that such instantiation always exists in Pr. Constraints of this form are referred to as **postponed constraints**.

A direct consequence of following this approach is that rule $(\stackrel{?}{\approx}\mathbf{XY})$ from the matching algorithm in Definition 5.2.6 becomes obsolete because, either the form of the unification constraint fits the definition of postponed constraint or is a clashing equality with distinct moderated variables belonging to the set $V_{RHS}(Pr)$ for some simple matching problem Pr. Either way, rule $(\stackrel{?}{\approx}\mathbf{XY})$ is not applicable, postponed constraints are delayed and rule $(\stackrel{?}{\approx}\bot)$ reduces clashing equalities from Definition 3.2.2 to $\varnothing$. Hence, rule $(\stackrel{?}{\approx}\mathbf{XY})$ is discarded from the simple-matching algorithm given in Definition 5.4.6. Additionally, instantiating rule $(\stackrel{?}{\approx}\mathbf{Inst})$, also from Definition 5.2.6, must be modified to restrict the construction of

v-substitutions only to unification constraints where the pattern is a moderated variable with trivial a-substitutions which does not belong to the set of variable symbols occurring on matched terms.

Prior to the update of the matching algorithm with the modifications suggested above, we define a notion of pattern matching where the pattern is a simple term. The *simple pattern matching problem* will be applied in the following chapter when mechanising extended nominal rewriting.

Definition 5.4.4 (Simple pattern matching problem)**.** A **simple pattern matching problem** consists of a pair of extended terms-in-context $\nabla \vdash l$, $\Delta \vdash s$ where l is a simple term and $V(\nabla \vdash l) \cap V(\Delta \vdash s) = \varnothing$. A simple pattern matching problem is represented as

$$(\nabla \vdash l) \overset{s}{?}\approx (\Delta \vdash s)$$

and interpreted as $\{l \overset{s}{?}\approx s\} \cup \nabla$.

We soon show how the term-in-context on the LHS of $\overset{s}{?}\approx$ belongs to a rewrite rule pattern whereas the RHS belongs to the matched term-in-context.

Intuitively, when pattern matching a (simple) problem $(\nabla \vdash l) \overset{s}{?}\approx (\Delta \vdash s)$ one wants to find a unifier σ such that $\mathscr{D}om(\sigma) \subseteq V(\nabla \vdash l)$ and $\Gamma \vdash l\sigma \approx_\alpha s$ and also $\Delta \vdash \nabla\sigma$ for all $\Gamma \in \mathbf{F}$, that is, $(\mathbf{F}, \sigma) \in \mathscr{U}(l \mathrel{?}\approx_? s, \nabla)$ following Definition 4.1.8. However, since we are about to introduce a matching algorithm for simple matching problems in Section 5.4.2, one can already make a direct correspondence between the solution to such matching algorithm and the solution to the simple pattern matching problem. This does not affect the satisfiability conditions stated above. Properties of the matching algorithm for simple matching problems can be found in Section 5.4.3. We continue the analysis after the formal definition.

Definition 5.4.5 (Solution to a simple pattern matching problem)**.** A **solution** to a simple pattern matching problem $(\nabla \vdash l) \overset{s}{?}\approx (\Delta \vdash s)$ is a v-substitution σ such that:

- $(\mathbf{F}, \sigma) \in \langle \{l \mathrel{?}\approx_? s\} \cup \nabla \rangle_{sol_{\overset{?}{\approx}}}$ and
- $\exists \Gamma \in \mathbf{F} : \Delta \vdash \Gamma$

where $\langle \{l \mathrel{?}\approx_? s\} \cup \nabla \rangle_{sol_{\overset{?}{\approx}}}$ is the normal form of $\{l \overset{s}{?}\approx s\} \cup \nabla$ by application of the simple-matching algorithm (see Theorem 5.4.11). We say that σ **solves** the pattern matching problem.

Informally, a solution to a simple pattern matching problem, $(\nabla \vdash l) \overset{s}{?}\approx (\Delta \vdash s)$, is a unifier σ from a pair $(\mathbf{F}, \sigma)$ if there is some $\Gamma \in \mathbf{F}$ derivable from Δ, $\Delta \vdash \Gamma$, and possibly some freshness context Δ^ϕ fresh for $\nabla \vdash l, s$ (i.e., $A(\Delta^\phi) \cap A(\nabla \vdash l, s) = \varnothing$ and $V(\Delta^\phi) \subseteq V(\nabla \vdash l, s)$)

which we have left implicit, and where $(\mathbf{F}, \sigma)$ belongs to the normal form of $\{l \overset{s}{?}\approx s\} \cup \nabla$ by application of the simple-matching algorithm as given in Theorem 5.4.11, $\langle \{l \,?\approx_? s\} \cup \nabla \rangle_{sol_{?\approx}}$ (in fact, we show the normal form is a singleton and thus the pair $(\mathbf{F}, \sigma)$ is unique). As a result of the simple-matching algorithm being a restrictive version of the general matching algorithm, $(\mathbf{F}, \sigma)$ satisfies $\forall \Gamma \in \mathbf{F}.\Gamma \vdash \{l\sigma \approx_\alpha s\sigma\} \cup \nabla\sigma$ and also $\forall \Gamma \in \mathbf{F}.\Gamma \vdash (\sigma \bullet \sigma) \approx_\alpha \sigma$ where $s\sigma = s$ because of the condition to matching constraints $V(l) \cap V(s) = \varnothing$ (see Definition 5.1.1) and then $\mathscr{D}om(\sigma) \subseteq V(\nabla \vdash l)$ by the syntactic restrictions imposed on the simple-matching algorithm (see Definition 5.4.6).

Now we are ready to apply the modifications described in this section to the general-matching algorithm to guarantee that solvable simple matching problems enjoy the property of unique most general unifier.

5.4.2 A matching algorithm for simple matching problems

Below, we present a matching algorithm for simple matching problems and call it *simple-matching algorithm* to distinguish it from the matching algorithm introduced in Definition 5.2.6. The simple-matching algorithm is built on top of the matching algorithm by applying further restrictions to the set of matching rules and providing an strategy to simplify the solution set so that a most general solution emerges, when one exists.

Definition 5.4.6 (Simple-matching algorithm)**.** Call **Simple-matching algorithm** the two-phase matching algorithm depicted in Definition 5.2.6 where rule $(?\approx\mathbf{XY})$ has been discarded from the set of rules and rule $(?\approx\mathbf{Inst})$ has been replaced by rule $(?\approx\mathbf{Inst_s})$ as follows:

$$(?\approx\mathbf{Inst_s}) \quad (\{\pi{\cdot}X \,?\approx_? t\} \cup Pr, \theta) \overset{X\notin Xs}{\Longrightarrow} (Pr[X \mapsto \pi^{-1}{\cdot}t], \theta \bullet [X \mapsto \pi^{-1}{\cdot}t])\}) \quad (\text{where } t \neq \phi'^{\frown}\pi'{\cdot}X)$$

Function $\langle Pr \rangle_{\mathtt{nf}_{?\approx}}$ is now the normal form of a matching problem Pr by application of the set of updated unification rules above. Additionally, the reduction strategy is extended with a priority on the application of matching rules where application of rule $(?\approx\mathbf{Inst_s})$ has the highest priority and application of rule $(?\approx\mathbf{X})$ has the lowest priority in the set. The remaining rules have the same priority.

Rule $(?\approx\mathbf{Inst_s})$ has been substituted directly by the instantiating rule for non-extended terms (see Urban et al., 2004, Figure 3). Such replacement is possible due to, following the notation above, $\mathtt{Cap}(\mathtt{t}, \varnothing) = \{\mathtt{t}\}$ since $\pi{\cdot}X$ has `Id` a-substitutions, thus generating a single v-substitution $[X \mapsto \pi^{-1}{\cdot}t]$. We could have preserved the use of function `Cap` by adjusting

the parameters of rule ($_?\approx$**Inst**). However, the use of the non-extended rule for variable instantiation reduces to one the number of computational steps to obtain the normal form of a unification constraint of form $\pi{\cdot}X \;_?\approx_?\; t$, avoiding a function call to `Cap` and subsequent application of rule ($_?\approx\equiv$) to reduce the generated constraint of form $t \;_?\approx_?\; t$.

We have imposed a precedence of application on the set of matching rules; the priority imposed to rule ($_?\approx$**Inst**$_\mathbf{s}$) over the rest of the unification rules forces the algorithm to generate v-substitutions as soon as they are readily available. On the other hand, rule ($_?\approx$**X**) has the lowest precedence of application in the unification set in order to delay the ramification of distinct sequences of reduction until any other unification constraint has been resolved. Then, solving rule ($_?\approx$**X**) is reduced to solve rule ($\approx_\alpha$**X**) when solving a simple matching problem since no variables are left to be instantiated (recall that RHS variables are seen as constants with suspended atom actions and there is always one fixed occurrence of each LHS variable symbol with trivial a-substitution as stated in Remark 5.4.3). As a result, each distinct solution $(\mathbf{F}_i, \sigma)$ from the solution set W shares the same unifier, σ, and, by application of $[\cdot]$ to W during phase three, the solution set is reduced to singleton $[W] = \{(\bigcup_{i\in \mathrm{I}} \mathbf{F}_i, \sigma)\}$ for $i \in \mathrm{I}$. We demonstrate this claim in Theorem 5.4.11.

5.4.3 Properties of the simple-matching algorithm

Next we show that the modifications and additional restrictions to the set of unification transitions from the general-matching algorithm do not affect the property of termination.

Lemma 5.4.7 (Termination)**.** *There is no infinite sequence of unification transitions for any matching problem Pr when applying the simple-matching algorithm given in Definition 5.4.6.*

Proof. It follows directly by Lemma 5.3.1 and the fact that postponed constraints do not pattern-match with any unification rule because of discarding rule ($_?\approx$**XY**) and the restriction on rule ($_?\approx$**Inst**$_\mathbf{s}$) so that a-substitutions have to be trivial for the rule to be applied. Therefore, postponed constraints terminate.

The order of priority imposed to the set of unification rules does not affect the property of termination. Therefore the result follows. ■

Next, we provide a function definition for the three-phase simple-matching algorithm.

Definition 5.4.8 (Three-phase simple-matching algorithm)**.** Write $\mathtt{Match}_s(Pr, V_{RHS}(Pr))$ for the normal form of matching problem *Pr* by means of the simple-matching algorithm given in Definition 5.4.6. Then, $\langle Pr\rangle_{sol_{?\approx}}$ is the result of applying function $[\cdot]$ from Definition 5.3.4 to set $\mathtt{Match}_s(Pr, V_{RHS}(Pr))$.

The definition of solution to both general and simple matching problems differs uniquely on the kind of matching problem at hand. The remark below states this fact.

Remark 5.4.9. A simple matching constraint of form $s \overset{s}{?}\approx t$ is checked by application of function $\langle\cdot\rangle_{sol_{?\approx}}$ given in Definition 5.4.8. Any other matching constraint of form $s \;?\approx t$ is checked by application of function $\texttt{Match}(\cdot,\cdot)$ from Definition 5.3.3. This extends to matching problems too.

An example of application of the three-phase strategy simple-matching algorithm follows.

Example 5.4.10. The simple matching problem with signature as in Example 2.1.1

$$Pr = \{\texttt{cons}([\texttt{a} \mapsto \texttt{H}]\cdot\texttt{F}, \texttt{map}([\texttt{a}]\texttt{F}, \texttt{H})) \overset{\texttt{s}}{?}\approx \texttt{cons}([\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, \texttt{map}([\texttt{b}]\texttt{X}, (\texttt{c d})\cdot\texttt{T}))\}$$

resolves to $\{(\{\{\texttt{a\#X}, \texttt{b\#X}\}, \{\texttt{a\#X}, \texttt{c\#T}, \texttt{d\#T}\}\}, [\texttt{H} \mapsto (\texttt{c d})\cdot\texttt{T}; \texttt{F} \mapsto (\texttt{a b})\cdot\texttt{X}])\}$ by application of function $\langle\cdot\rangle_{sol_{?\approx}}$ from Definition 5.4.8 to Pr as follows.

Phase 1:

$\{\texttt{cons}([\texttt{a} \mapsto \texttt{H}]\cdot\texttt{F}, \texttt{map}([\texttt{a}]\texttt{F}, \texttt{H})) \;?\approx_? \texttt{cons}([\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, \texttt{map}([\texttt{b}]\texttt{X}, (\texttt{c d})\cdot\texttt{T}))\}, \texttt{Id})$
$\Longrightarrow_{(?\approx\textbf{f})} \{([\texttt{a} \mapsto \texttt{H}]\cdot\texttt{F}, \texttt{map}([\texttt{a}]\texttt{F}, \texttt{H})) \;?\approx_? ([\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, \texttt{map}([\texttt{b}]\texttt{X}, (\texttt{c d})\cdot\texttt{T}))\}, \texttt{Id})$
$\overset{(?\approx\textbf{tupl})}{\Longrightarrow} \{[\texttt{a} \mapsto \texttt{H}]\cdot\texttt{F} \;?\approx_? [\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, \texttt{map}([\texttt{a}]\texttt{F}, \texttt{H}) \;?\approx_? \texttt{map}([\texttt{b}]\texttt{X}, (\texttt{c d})\cdot\texttt{T})\}, \texttt{Id})$
$\Longrightarrow_{(?\approx\textbf{fun})} \{[\texttt{a} \mapsto \texttt{H}]\cdot\texttt{F} \;?\approx_? [\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, ([\texttt{a}]\texttt{F}, \texttt{H}) \;?\approx_? ([\texttt{b}]\texttt{X}, (\texttt{c d})\cdot\texttt{T})\}, \texttt{Id})$
$\Longrightarrow_{(?\approx\textbf{tupl})} \{[\texttt{a} \mapsto \texttt{H}]\cdot\texttt{F} \;?\approx_? [\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, [\texttt{a}]\texttt{F} \;?\approx_? [\texttt{b}]\texttt{X}, \texttt{H} \;?\approx_? (\texttt{c d})\cdot\texttt{T}\}, \texttt{Id})$
$\overset{H\notin\{T,X\}}{\Longrightarrow}_{(?\approx\textbf{Inst}_\textbf{s})} (\{[\texttt{a} \mapsto (\texttt{c d})\cdot\texttt{T}]\cdot\texttt{F} \;?\approx_? [\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, [\texttt{a}]\texttt{F} \;?\approx_? [\texttt{b}]\texttt{X}\}, \texttt{Id} \bullet \theta)$
where $\theta = [\texttt{H} \mapsto (\texttt{c d})\cdot\texttt{T}]$
$\Longrightarrow_{(?\approx[\textbf{b}])} \{[\texttt{a} \mapsto (\texttt{c d})\cdot\texttt{T}]\cdot\texttt{F} \;?\approx_? [\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, \texttt{a\#X}, \texttt{F} \;?\approx_? (\texttt{a b})\cdot\texttt{X}\}, \texttt{Id} \bullet \theta)$
$\overset{F\notin\{T,X\}}{\Longrightarrow}_{(?\approx\textbf{Inst}_\textbf{s})} (\{[\texttt{a} \mapsto (\texttt{c d})\cdot\texttt{T}]\hat{}(\texttt{a b})\cdot\texttt{X} \;?\approx_? [\texttt{b} \mapsto \texttt{T}]\cdot\texttt{X}, \texttt{a\#X}\}, \texttt{Id} \bullet \theta \bullet \theta')$
where $\theta' = [\texttt{F} \mapsto (\texttt{a b})\cdot\texttt{X}]$
$\Longrightarrow_{(?\approx\textbf{X})} (\{\texttt{a\#X}, \texttt{b\#X}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup (\{\texttt{a\#X}, (\texttt{c d})\cdot\texttt{T} \;?\approx_? \texttt{T}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup$
$(\{\texttt{b} \;?\approx_? \texttt{a}, \texttt{b\#X}, \texttt{a\#X}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup (\{\texttt{b} \;?\approx_? \texttt{a}, (\texttt{c d})\cdot\texttt{T} \;?\approx_? \texttt{T}, \texttt{a\#X}\}, \texttt{Id} \bullet \theta \bullet \theta')$
where $\texttt{diffSet}([\texttt{a} \mapsto (\texttt{c d})\cdot\texttt{T}]\hat{}(\texttt{a b}), [\texttt{b} \mapsto \texttt{T}], \varnothing)_{\texttt{X}}^{\{\texttt{a,b}\}} = \{\{\texttt{a\#X}, \texttt{b\#X}\}, \{\texttt{a\#X}, (\texttt{c d})\cdot\texttt{T} \;?\approx_? \texttt{T}\},$
$\{\texttt{b} \;?\approx_? \texttt{a}, \texttt{b\#X}\}, \{\texttt{b} \;?\approx_? \texttt{a}, (\texttt{c d})\cdot\texttt{T} \;?\approx_? \texttt{T}\}\}$
$\Longrightarrow^*_{(?\approx\perp)} (\{\texttt{a\#X}, \texttt{b\#X}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup (\{\texttt{a\#X}, (\texttt{c d})\cdot\texttt{T} \;?\approx_? \texttt{T}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup \varnothing$
$\Longrightarrow_{(?\approx_?\textbf{X})} (\{\texttt{a\#X}, \texttt{b\#X}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup (\{\texttt{a\#X}, \texttt{c\#T}, \texttt{d\#T}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup$
$(\{\texttt{a\#X}, \texttt{c\#T}, \texttt{c} \;?\approx_? \texttt{d}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup (\{\texttt{a\#X}, \texttt{d} \;?\approx_? \texttt{c}, \texttt{d\#T}\}, \texttt{Id} \bullet \theta \bullet \theta') \cup$
$(\{\texttt{a\#X}, \texttt{d} \;?\approx_? \texttt{c}, \texttt{c} \;?\approx_? \texttt{d}\}, \texttt{Id} \bullet \theta \bullet \theta')$
where $\texttt{diffSet}((\texttt{c d}), \varnothing, \varnothing)_{\texttt{T}}^{\{\texttt{c,d}\}} = \{\{\texttt{c\#T}, \texttt{d\#T}\}, \{\texttt{c\#T}, \texttt{c} \;?\approx_? \texttt{d}\},$
$\{\texttt{d} \;?\approx_? \texttt{c}, \texttt{d\#T}\}, \{\texttt{d} \;?\approx_? \texttt{c}, \texttt{c} \;?\approx_? \texttt{d}\}\}$

$\Longrightarrow^{*}_{(?\approx\perp)} (\{\mathtt{a\#X}, \mathtt{b\#X}\}, \mathtt{Id} \bullet \theta \bullet \theta') \cup (\{\mathtt{a\#X}, \mathtt{c\#T}, \mathtt{d\#T}\}, \mathtt{Id} \bullet \theta \bullet \theta') \cup \varnothing$

Phase 2:

$\{(\langle\{\mathtt{a\#X}, \mathtt{b\#X}\}\rangle_{\mathtt{nf}}, [\mathtt{H} \mapsto (\mathtt{c}\ \mathtt{d})\cdot\mathtt{T}; \mathtt{F} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{X}]) \cup$
$(\langle\{\mathtt{a\#X}, \mathtt{c\#T}, \mathtt{d\#T}\}\rangle_{\mathtt{nf}}, [\mathtt{H} \mapsto (\mathtt{c}\ \mathtt{d})\cdot\mathtt{T}; \mathtt{F} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{X}])\}$

Phase 3:

$[\{(\langle\{\mathtt{a\#X}, \mathtt{b\#X}\}\rangle_{\mathtt{nf}}, [\mathtt{H} \mapsto (\mathtt{c}\ \mathtt{d})\cdot\mathtt{T}; \mathtt{F} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{X}]) \cup (\langle\{\mathtt{a\#X}, \mathtt{c\#T}, \mathtt{d\#T}\}\rangle_{\mathtt{nf}},$
$[\mathtt{H} \mapsto (\mathtt{c}\ \mathtt{d})\cdot\mathtt{T}; \mathtt{F} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{X}])\}] =$
$\{(\{\{\mathtt{a\#X}, \mathtt{b\#X}\}, \{\mathtt{a\#X}, \mathtt{c\#T}, \mathtt{d\#T}\}\}, [\mathtt{H} \mapsto (\mathtt{c}\ \mathtt{d})\cdot\mathtt{T}; \mathtt{F} \mapsto (\mathtt{a}\ \mathtt{b})\cdot\mathtt{X}])\}.$

Next, we formalise the fact that the application of the simple-matching algorithm to a simple matching problem resolves into a set of solutions sharing the same unifier, if the problem is unifiable. Then, by the merging of solutions via function $[\cdot]$, the three phased-strategy matching algorithm returns a unique solution to the simple matching problem. This theorem is the main result of this chapter.

Theorem 5.4.11 (Normal form of a simple matching problem)**.** *Given a simple matching constraint $s \overset{s}{?}\!\approx t$ as defined in Definition 5.4.1, then either $\langle s \ {}_?\!\approx_? t\rangle_{sol_{?^{\approx}}} = [\mathtt{Match}_s(s \ {}_?\!\approx_? t, V_{RHS}(\{s \ {}_?\!\approx_? t\}))] = \{(\mathbf{F}, \theta)\}$ where $\mathbf{F}$ is a collection of freshness contexts and θ is an idempotent unifier or $\langle s \ {}_?\!\approx_? t\rangle_{sol_{?^{\approx}}} = \varnothing$.*

As a corollary, the normal form of a simple matching problem Pr is also of form $\{(\mathbf{F}, \theta)\}$ $= [\mathtt{Match}_s(Pr, V_{RHS}(Pr))] = \langle Pr\rangle_{sol_{?^{\approx}}}$, if Pr is unifiable. Otherwise, $\langle Pr\rangle_{sol_{?^{\approx}}} = \varnothing$.

Proof. For the first claim, by Lemma 5.4.7 we know that the simple-matching algorithm terminates and by Lemma 5.3.7 and Remark 5.4.3 we also know the normal form of $\mathtt{Match}_s(s \ {}_?\!\approx_? t, V_{RHS}(\{s \ {}_?\!\approx_? t\}))$ has the structure of a set of solutions if $\{s \ {}_?\!\approx_? t\}$ is unifiable by means of simple-matching or otherwise $\mathtt{Match}_s(s \ {}_?\!\approx_? t, V_{RHS}(\{s \ {}_?\!\approx_? t\})) = \varnothing$.

Now, one has to show that, if $\mathtt{Match}_s(s \ {}_?\!\approx_? t, V_{RHS}(\{s \ {}_?\!\approx_? t\})) \neq \varnothing$ then, the application of $[\cdot]$ to $\mathtt{Match}_s(s \ {}_?\!\approx_? t, V_{RHS}(\{s \ {}_?\!\approx_? t\}))$ reduces the set to a singleton and we are done. This is proved as follows.

There is only one matching rule in the simple-matching algorithm that leads to non-determinism of results, namely, rule $({}_?\!\approx\mathbf{X})$. Then, if t is ground, the result follows trivially. Otherwise, by the restriction imposed on simple matching constraints in Definition 5.4.1 we have at least one fixed variable occurrence having trivial a-substitutions for each variable symbol occurring on s. Then, the reduction strategy we have applied to the set of matching

rules in Definition 5.4.6 prioritises the generation of v-substitutions by rule $(_?\approx\mathbf{Inst_s})$ and delays the application of rule $(_?\approx\mathbf{X})$ until no further unification constraints are left in the problem. Therefore, we are left to prove that rule $(_?\approx\mathbf{X})$ produces candidate solutions sharing α-equivalent unifiers. This is the case since application of rule $(_?\approx\mathbf{X})$ has the lowest priority, therefore it is a fact that no further applications of rule $(_?\approx\mathbf{Inst_s})$ are possible since the variable symbols left in the remaining problem belong to set $V(t)$ and therefore treated as constants by means of the side conditions in $(_?\approx\mathbf{Inst_s})$. Hence application of rule $(_?\approx\mathbf{X})$ is reduced to application of rule $(\approx_\alpha\mathbf{X})$ and then we are able to claim that all candidate solutions returned by such rule contain structurally equal v-substitution compositions, that is, $\mathtt{Match}_s(s\ {}_?\approx_?\ t, Vf(\{s\ {}_?\approx_?\ t\})) = \{\{(\mathbf{F}_1,\sigma),\ldots,(\mathbf{F}_n,\sigma)\}\}$ and the result follows by application of function $[\cdot]$.

The corollary follows by the claim above and Lemma 5.3.7. ■

5.5 Conclusion

In this chapter, the matching problem for extended terms was characterised and a decidable matching algorithm was provided. This first approach to the matching process is not unitary, returning a set of solutions for the general case. However, it is useful to provide an operational definition of closedness for extended terms to find out whether there exists any solution showing that some given extended term is closed. Properties and examples for this algorithm were also described.

Then, a class of matching constraints that we call simple was described where matching is unitary. Constraints belonging to this class have at least one variable occurrence with trivial a-substitutions for each variable symbol occurring in the pattern term. As a result, a simple-matching algorithm was designed on top of the previous matching algorithm where matching constraints of form $\phi\hat{}\pi\cdot X\ {}_?\approx^s t$ are delayed until an instantiation for X is readily available from another occurrence of X with trivial a-substitutions. Such occurrence always exists because of the conditions imposed on matching constraints. Then, the chapter was concluded by showing that the simple-matching algorithm is unitary for the class of simple matching problems.

Now, we are ready to provide a definition of nominal rewriting in the presence of a-substitutions.

Part IV

Extended Rewriting and Applications

Chapter 6

Extended Nominal Rewriting

This chapter aims to provide a notion of closed rewriting suitable for the extended formalism with a view to the enhancement of the translation algorithms given in (Domínguez and Fernández, 2014) and (Domínguez and Fernández, 2015) for the translation of NRSs to CRSs and the translation of CRSs to NRSs respectively.

Nominal rewriting (Fernández and Gabbay, 2007) generalises the first-order term rewriting framework by adding a binding mechanism to first-order syntax trees, following *the nominal sets approach* (Pitts, 2003; Urban et al., 2004) and a formal notion of freshness. Extended nominal rewrite systems generalise further by introducing substitution at the level of terms as a primitive theory which is not relegated to the meta-level but dealt with explicitly, internalized as a logical derivation via atom swaps and the freshness relation.

One of the main properties of first-order rewriting which extends to the nominal framework is that *rule patterns* are sensitive to distinction among names, be they atoms or variables symbols, rather than to the particular names chosen. This notion of invariance up to permuted renaming of atoms was already introduced in the seminal paper for nominal sets (Gabbay and Pitts, 2002) as a fundamental property of atoms, and further developed in (Pitts, 2003) for nominal logic, stating that the satisfiability of axioms for nominal logic is guaranteed under atom swapping. In nominal rewriting theories this is known as *meta-level equivariance* or *equivariance*, to distinguish it from the internal equivariance of predicates $\approx_\alpha$ and # as shown previously in Corollary 3.1.5 (see Chapter 3). Meta-level equivariance presents a technical problem when automating the matching of rule patterns to terms in nominal rewriting as already denoted by Cheney in (Cheney, 2004a), namely that the standard nominal unification algorithm introduced in (Urban et al., 2004) does not take into account equivariance and thus rewriting is incomplete as a result: any collection of rewriting rules $l \to r$ used to define a relation $\to$ in nominal rewriting is equivalent to $(a\ b)\cdot l \to (a\ b)\cdot r$, for any pair of atoms a, b, because of the equivariance property, however, when l, r are considered

standard nominal terms as defined in (Urban et al., 2004) or extended terms as introduced in Chapter 2, free atoms a, b occurring in l, r are logically equivalent if and only if they are equal as terms (Cheney, 2010). This issue also directly affects our simple-matching algorithm presented in 5.4.8.

In (Cheney, 2010), Cheney develops an intricate unification algorithm (its decision procedure is shown to be **NP**-complete) for nominal logic that takes into account the equivariance property, producing a complete set of finitely many solutions. Additionally in (Cheney, 2010), there is also a polynomial time equivariant matching algorithm for a class of constraints where swappings on the LHS term are trivial, that is, `Id` swappings only. Fernández and Gabbay (Fernández and Gabbay, 2007) followed a distinct approach, taking equivariance as an implicit property in rule patterns. This was achieved by considering rules modulo permutative renamings of atoms and working with the *equivariant closure* of the set of rules. By following this approach, matching a nominal term with an equivariant rule was kept as a quadratic decision problem instead (see (Calvès, 2010) and (Levy and Villaret, 2010)).

The machinery for rewriting in the extended framework follows the path specified in (Fernández and Gabbay, 2007), assimilating equivariance conveniently within the definition of a rewrite system so that the unification algorithm described in 5.2.6 can be employed as a matching tool, producing unique principal solutions during rewriting in the extended framework.

In this chapter, we aim to define a model of nominal rewriting suitable for our extended framework, reviewing and adapting results and techniques available from the non-extended model designed in (Fernández and Gabbay, 2007). As a result, two formulations of extended nominal rewriting are presented, one more expressive, using equivariant matching over sets of *simple rewrite rules*, and one more efficient by use of *closed-simple matching* and *closed-simple rewrite rules* which nevertheless encompasses most scenarios of interest.

6.1 Extended Rewrite Rules

In order to derive the rewrite relation one must specify a *rewrite system*, consisting of *rewrite rules* over a signature Σ. Additionally, a notion of matching terms-in-context is required to mechanize the process of rewriting using a given set of rewrite rules. In the previous chapter we defined a unitary matching algorithm for a subset of matching constraints, known as simple matching constraints. The notion of a simple term is instrumental in the extended formalism when a-substitutions are allowed on terms occurring on the LHS of a rewrite rule. Simple pattern matching problems (see Definition 5.4.4) enjoy most general unifiers, if solvable, when matching with the three phase simple-matching algorithm given in

Definition 5.4.8. As a result, *simple extended nominal rewrite rules* preserve the property of determinism of matching when generating a rewrite step at a given position as well as well as preserving the property of introducing no new variables during rewriting. This is justified in Section 6.2, when defining the rewriting process for eNRSs.

First, we begin by providing a definition of simple rewrite rule for the extended nominal formalism.

Definition 6.1.1. A **simple extended nominal rewrite rule**, or just **simple (rewrite) rule**, over a signature Σ is a tuple (∇, l, r), written $R = (\nabla \vdash l \to r)$, where

- ∇ is a consistent freshness context,
- l and r are extended nominal terms such that l is simple as in Definition 5.4.1 and $V(\nabla, r) \subseteq V(l)$. We refer to the LHS rule term as **pattern** (of simple rule R) or **simple rule pattern** (of R).

We may write $l \to r$ for $\varnothing \vdash l \to r$.

Freshness contexts on rewrite rules offer the possibility of restricting matching solutions between a term-in-context and a simple rule pattern by specifying freshness conditions that must be satisfied by any solution to the simple pattern matching problem. This is formalised later, when providing a means of inducing rewrite steps on the class of extended nominal terms over a signature and a rewrite system.

The second condition states that variable symbols in RHS of simple rules or in its corresponding freshness context must also occur in rule patterns and that rule patterns have to be simple, that is, there is at least one occurrence of a variable symbol in the pattern that has a fixed position and suspended trivial a-substitutions. Then, RHS variable symbols occur in the pattern at a fixed position. This avoids the introduction of fresh variables during rewriting since v-substitutions are generated for each variable symbol occurring in a simple rule pattern. Alternatively, by restricting a-substitutions to terms on the right of rules solely, one is able to recover the conditions from standard nominal rewriting, that is, the condition for the rule pattern to be simple is dropped from the definition.

In the sequel, we refer to simple rewrite rules and simple rule patterns simply as rewrite rules and patterns since they are the only class of extended rewrite rules we work with throughout this chapter. Also, when providing examples, we work with a shallow embedding so that atom substitution can be directly applied without the need of a rewrite rule for function application.

Rewrite rules are instantiated as follows:

$$(\nabla \vdash l \to r)[X \mapsto t] \triangleq l[X \mapsto t] \to r[X \mapsto t] \text{ where } \langle \nabla[X \mapsto t] \rangle_{\mathtt{nf}} \neq \bot$$

by application of Definition 2.3.9 and Definition 4.1.6. The collection of freshness contexts $\langle \nabla[X \mapsto t] \rangle_{\mathtt{nf}}$ provides soundness of the rewrite relation by checking that v-substitution $[X \mapsto t]$ satisfies freshness context ∇, namely, $\langle \nabla[X \mapsto t] \rangle_{\mathtt{nf}} \neq \bot$, and it is not maintained during the rewriting process.

Below are some examples of rewrite rules in the extended framework.

Example 6.1.2.

- $R_\pi = \mathtt{par}(\mathtt{out}(\mathtt{a},\mathtt{b}),\mathtt{in}(\mathtt{a},[\mathtt{c}]\mathtt{P})) \to [\mathtt{c} \mapsto \mathtt{b}]\cdot\mathtt{P}$ is a rewrite rule with signature as in Example 2.1.1, inspired from (Fernández and Gabbay, 2005), representing communication of data through channels in the π calculus.
- $\mathtt{a}\#\mathtt{X} \vdash \mathtt{X} \to \mathtt{abs}([\mathtt{a}]\mathtt{app}(\mathtt{X},\mathtt{a}))$ with signature as in Example 1.2.1 represents the η-expansion rewrite rule of the λ-calculus with names.
- $\mathtt{map}([\mathtt{a}]\mathtt{F},\mathtt{cons}(\mathtt{H},\mathtt{T})) \to \mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F},\mathtt{map}([\mathtt{a}]\mathtt{F},\mathtt{T}))$ is the rewrite rule with signature as in Example 2.1.1, containing the recursive case of the higher-order function `map`. Observe that we could have added a unary function symbol `lam` to emphasise that $[\mathtt{a}]\mathtt{F}$ represents a function, namely $\mathtt{lam}[\mathtt{a}]\mathtt{F}$, however it is not necessary due to the inclusion of a-substitutions. More generally, by extending nominal rewriting with a-substitutions we do not need to include function symbols `lam` and `app` to denote lambda-abstraction and function application respectively. Accordingly, β-reduction steps are no longer needed for this and other similar higher-order function rules. In this case, one can simply write $[\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}$ to apply $[\mathtt{a}]\mathtt{F}$ to $\mathtt{H}$.
- $\mathtt{a} \to \mathtt{a}$ is a valid rule. Soon we show how our notion of rewriting deals with matching of unabstracted atoms.
- $[\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X} \to \mathtt{Y}$ is not a rewrite rule because $([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X})$ is not simple.

It is commonplace in classic term rewriting for rewrite rules to be sensitive to distinction among variable symbols, but unaffected by the choice of symbols. This property also applies to extended nominal rewriting. Since nominal syntax distinguishes between object-level variables and meta-variables, the property affects both. As a result, and in order to generate the rewrite relation, rewrite rules must be considered up to renaming of variable symbols as well as to permuted renaming of atoms. Therefore, if $R = a\#Y \vdash \mathtt{app}(\mathtt{lam}[a]X,(a,Y)) \to [a \mapsto (a,Y)]\cdot X$ is a rewrite rule in a given rewrite system, $a\#X \vdash \mathtt{app}(\mathtt{lam}[a]Y,(a,X)) \to [a \mapsto (a,X)]\cdot Y$ is seen as a *variant of* R, and so is $b\#Y \vdash \mathtt{app}(\mathtt{lam}[b]X,(b,Y)) \to [b \mapsto (b,Y)]\cdot X$. Notice that the last rule variant is derived by application of swapping $(a\ b)$ on the rewrite rule R. However, observe how swappings do not suspend on variables when providing permuted variants of rules, they only affect atoms, unlike the permutation action in Definition 2.1.8.

This is due to the property of equivariance of nominal logic (Pitts, 2003) which was already taken into account when inducing the rewrite relation in NRSs (Fernández and Gabbay, 2007).

Equivariance is here a meta-level property (meta-level equivariance), that is, it is necessary to distinguish meta-level equivariance from the internal property of equivariance proved in Lemma 2.5.5 for predicates # and $\approx_\alpha$. The following definition extends the meta-permutation action to extended terms.

Definition 6.1.3 (Meta-permutation action)**.** Let t be an extended nominal term and π a meta-permutation. Then, the **meta-application** of π on t, t^π, is inductively defined as follows:

$$a^\pi \triangleq \pi(a) \quad ([a]t)^\pi \triangleq [a^\pi]t^\pi \quad (ft)^\pi \triangleq ft^\pi \quad (s_1,\ldots,s_n)^\pi \triangleq (s_1^\pi,\ldots,s_n^\pi)$$
$$(\phi\hat{}\pi_1{\cdot}X)^\pi \triangleq \phi^\pi\hat{}\pi_1^\pi{\cdot}X \quad \text{where } \pi_1^\pi \triangleq \begin{cases} (a^\pi\ b^\pi)\circ\tau^\pi, & \text{if } \pi_1 = (a\ b)\circ\tau \\ \mathtt{Id}, & \text{if } \pi_1 = \mathtt{Id} \end{cases}$$
$$\text{and } \phi^\pi \triangleq [\pi(a_1)\mapsto s_1^\pi;\ldots;\pi(a_n)\mapsto s_n^\pi] \text{ for } \phi = [a_1\mapsto s_1;\ldots;a_n\mapsto s_n].$$

In the sequel, we sometimes refer to meta-permutations as permutations when there is no uncertainty of meaning.

Example 6.1.4. If $t = [a][b\mapsto[a]a]\hat{}(a\ c){\cdot}X$ then $t^{(a\ b)} = [b][a\mapsto[b]b]\hat{}(b\ c){\cdot}X$.

Composition of meta-permutations extends to terms.

Lemma 6.1.5. *For any given term t and permutations π,π' we have $t^{(\pi'\circ\pi)} = t^{\pi^{\pi'}}$.*

Proof. By induction on the syntax of t, using Definition 6.1.3. We omit the inductive proof. ■

The meta-permutation application extends naturally to freshness contexts and rewrite rules, namely

$$\nabla^\pi = \{\pi(a)\#X \mid a\#X \in \nabla\}$$

such that

$$(\nabla \vdash l \to r)^\pi = \nabla^\pi \vdash l^\pi \to r^\pi.$$

In the presence of v-substitutions, permutations $\pi{\cdot}$- and meta-permutations -$^\pi$ are interdefinable as it was shown in this technical lemma from (Fernández and Gabbay, 2007, Lemma 41). It will be used later in Theorem 6.2.10.

Lemma 6.1.6. *Given a freshness context Δ, nominal term s and v-substitution σ that maps each X that occurs in s to $\pi{\cdot}X$. Then $\Delta \vdash \pi{\cdot}s \approx_\alpha s^\pi\sigma$.*

Proof. By induction on the structure of s.

The case for $s = \phi\hat{}\pi_1 \cdot X$ follows from Lemma 41 (Fernández and Gabbay, 2007), Lemma 2.5.14 and the inductive hypothesis where we have $\Delta \vdash \pi \cdot (\phi(a)) \approx_\alpha (\phi(a))^\pi \sigma$ for each $a \in \mathscr{D}om(\phi)$. Other cases have already been proved in Lemma 41 (Fernández and Gabbay, 2007). ■

The meta-permutation application allows us to define the set of permuted variant rules with respect to any given rewrite rule. We formalise this notion below.

Definition 6.1.7. (Equivariant closure) If R is a rewrite rule from a set of rewrite rules Rw, then so is R^π for any π. We define R^π as a **permuted variant** of R.

More generally, define the **equivariant closure** of Rw, written $\mathtt{clos}(Rw)$, as the closure of the set of rewrite rules Rw by application of π, that is, the set of all permuted variants of rewrite rules in Rw.

We are ready to specify the process of rewriting for the extended formalism.

6.2 Elementary Nominal Rewriting

Definition 6.2.1. A **rewrite system** $\mathscr{R}$ is a pair (Σ, Rw) of a signature Σ and a set Rw of rewrite rules over Σ.

Example 6.2.2. To represent the standard higher-order function *map*, we define the following set of rules over the signature given in Example 2.1.1:

$$\mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{nil}) \to \mathtt{nil};$$
$$\mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{cons}(\mathtt{H}, \mathtt{T})) \to \mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}] \cdot \mathtt{F}, \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{T})).$$

Example 6.2.3. To characterise the $\beta\eta$-reduction rewrite system of the λ-calculus with names over the signature given in Example 1.2.1, we define rules $(\beta), (\eta)$ which represent, respectively, β-reduction and η-reduction as follows:

$$(\beta) \quad \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{X}), \mathtt{Y}) \to [\mathtt{a} \mapsto \mathtt{Y}] \cdot \mathtt{X};$$
$$(\eta) \quad \mathtt{a\#X} \vdash \mathtt{lam}([\mathtt{a}]\mathtt{app}(\mathtt{X}, \mathtt{a})) \to \mathtt{X}.$$

The use of extended nominal terms in the example above reduces the verbosity of the syntax used to define a rewrite system of the λ-calculus with names. More precisely, the semantics of capture-avoiding a-substitutions have been relegated to the meta-level where capture-avoiding substitution of the λ-calculus is managed syntactically by our extended formalism, reducing the set of rewrite rules necessary to provide a nominal representation

of the rewrite system. As a comparison, in (Fernández et al., 2004) we find an alternative nominal rewrite system representing β- & η-reduction which uses an explicit substitution operator instead. The system contains five additional rewrite rules in order to simulate β-reduction over nominal syntax.

The rewrite relation on extended nominal terms is defined next.

A rewrite rule $R = (\nabla \vdash l \to r)$ determines a set of **rewrites** $R\sigma$ for all v-substitutions σ that satisfy ∇. We call an instance of l under σ a **redex**. A redex $l\sigma$ is replaced by its **contractum** $r\sigma$ inside a context u at a distinguished position p, $u[r\sigma]_p$, giving rise to a rewrite step over a given freshness context Δ if $\Delta \vdash \nabla\sigma$, namely, $\Delta \vdash u[l\sigma]_p \to u[r\sigma]_p$.

We provide next a formal description of the particularities of rewriting in the extended nominal framework. The notion of rewriting below specifies a generalisation of nominal rewriting in the presence of a-substitutions and follows the notation given in (Fernández and Gabbay, 2010) for the non-extended nominal formalism. We refer to this interpretation as **elementary rewriting**, as originally named in (Fernández et al., 2004), in order to distinguish it from a more elaborated notion of rewriting, which we introduce in Section 6.4, that builds on top of this one.

In the sequel, separate judgements $\Delta \vdash C_1, \ldots, \Delta \vdash C_n$ are displayed with notation $\Delta \vdash (C_1, \ldots, C_n)$

Definition 6.2.4.

- A rewrite system $\mathscr{R}$ induces a **one-step rewrite relation** $\Delta \vdash s \xrightarrow{R} t$ on terms s,t as follows: $\exists(\nabla \vdash l \to r) \in \mathscr{R}$, $p \in \mathscr{P}os(s)$, permutation π and v-substitution θ such that

$$\frac{\Delta \vdash (\nabla^{\pi}\theta, \quad s|_p \approx_\alpha l^{\pi}\theta, \quad s[r^{\pi}\theta]_p \approx_\alpha t)}{\Delta \vdash s \xrightarrow{R} t} \ (\to\textbf{Rew})$$

- The **(multi-step) rewrite relation** $\Delta \vdash_{\mathscr{R}} s \to t$ is the reflexive, transitive closure of the one-step rewrite relation over a system $\mathscr{R}$, namely, the least relation that includes the one-step rewrite relation and such that, for all Δ, s, t, u,
  - $\Delta \vdash_{\mathscr{R}} s \to t$ if $\Delta \vdash s \approx_\alpha t$;
  - $\Delta \vdash_{\mathscr{R}} s \to u$ if $\Delta \vdash_{\mathscr{R}} s \to t$ and $\Delta \vdash_{\mathscr{R}} t \to u$.

  If $\Delta \vdash_{\mathscr{R}} s \to t$ holds, we say that s **rewrites with** $\mathscr{R}$ **to** t **in the context** Δ. Otherwise we write $\Delta \nvdash_{\mathscr{R}} s \to t$ when it is not derivable.

- A **normal form** is a term-in-context which does not rewrite. A rewrite system $\mathscr{R}$ is **terminating** when there are no infinite rewriting sequences.

Both the one-step rewrite relation and the multi-step rewrite relation are derivable from some freshness context Δ and possibly a freshness context Δ^{ϕ} which we have left implicit. Freshness context Δ^{ϕ} is fresh for the rule from the rewrite system that induces the rewrite relation, that is, Δ^{ϕ} is fresh for $\nabla \vdash (l,r)$ if $\nabla \vdash l \to r$ is the rule applied in the premise of $(\to\mathbf{Rew})$. Δ^{ϕ} entails the rewrite relation for the case where suspended a-substitutions act on variable instances containing abstractions. As previously explained in Section 2.3, there is an infinite set of names we could have chosen to identify the abstractors since equality of nominal terms is modulo *provable* α, so it is only fair that we use any other available name to avoid atom capture.

In the design of the rewrite relation, prior initiating the rewriting process we have chosen some fresh set of primitive constraints Δ^{ϕ} large enough to entail the derivation of the multi-step rewrite relation. Accordingly, we adopt a design where the freshness context is fixed during rewriting, following the specification given for the standard counterpart in (Fernández and Gabbay, 2007). If ever a larger fresh Δ^{ϕ} was needed, the rewriting process would be initiated again generating a a larger set of fresh atoms to add as primitive constraints in Δ^{ϕ}. Alternatively, Δ^{ϕ} could be added as needed during the rewriting process. Both solutions are valid and the resulting rewrite steps α-equivalent, because of the (object-level) equivariance property stated on the second claim of Corollary 3.1.5. Then, an α-equivalent unifier could have been applied (see Lemma 2.5.13) so that Δ^{ϕ} is discarded (see strengthening lemma from Lemma 2.5.3). The use of some fresh context Δ^{ϕ} for the given rule is closely related to standard closed rewriting and completeness of rewriting for *equational reasoning* as given in (Fernández and Gabbay, 2010).

In the sequel, the notation $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ emphasises the fact that the one-step rewrite relation occurs with some specific rule R, position p, permutation π and v-substitution θ.

We continue by illustrating the notion of elementary rewriting with a few examples.

Example 6.2.5. The term-in-context $\vdash \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{lam}([\mathtt{b}]\mathtt{app}(\mathtt{a},\mathtt{b}))),\mathtt{b})$ rewrites to a normal form in one step with the rule (β) from the $\beta\eta$-reduction system in Example 6.2.3, at position ε with permutation $\mathtt{Id}$ and v-substitution $\theta = [\mathtt{X} \mapsto \mathtt{lam}([\mathtt{b}]\mathtt{app}(\mathtt{a},\mathtt{b}));\mathtt{Y} \mapsto \mathtt{b}]$ as follows,

$$\vdash \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{lam}([\mathtt{b}]\mathtt{app}(\mathtt{a},\mathtt{b}))),\mathtt{b}) \to_{\langle(\beta),\varepsilon,\mathtt{Id},\theta\rangle} \mathtt{lam}([\mathtt{c}]\mathtt{app}(\mathtt{b},\mathtt{c}))$$

Capture of unabstracted atom b has been avoided by the internal machinery of the extended nominal framework, unlike alternative nominal rewrite systems for the λ-calculus with names, e.g. (Fernández and Gabbay, 2010; Fernández et al., 2004), where semantics

of capture-avoiding substitution were added as rewrite rules in the system, increasing the number of steps to obtain a normal form.

Example 6.2.6. The term-in-context $\vdash \mathtt{par}(\mathtt{out}(\mathtt{b},\mathtt{c}),\mathtt{in}(\mathtt{b},[\mathtt{a}]\mathtt{in}(\mathtt{b},[\mathtt{d}](\mathtt{a},\mathtt{d})))$ with signature as in Example 2.1.1 for the asynchronous π-calculus rewrites in one step with rule R_π from Example 6.1.2, at position ε with permutation $(\mathtt{a}\ \mathtt{c})(\mathtt{a}\ \mathtt{b})$ and v-substitution $\theta = [\mathtt{P} \mapsto \mathtt{in}(\mathtt{b},[\mathtt{d}](\mathtt{a},\mathtt{d}))]$ as follows,

$$\vdash \mathtt{par}(\mathtt{out}(\mathtt{b},\mathtt{c}),\mathtt{in}(\mathtt{b},[\mathtt{a}]\mathtt{in}(\mathtt{b},[\mathtt{d}](\mathtt{a},\mathtt{d}))) \rightarrow_{\langle \mathtt{R}_\pi,\varepsilon,(\mathtt{a}\ \mathtt{c})(\mathtt{a}\ \mathtt{b}),\theta\rangle} \mathtt{in}(\mathtt{b},[\mathtt{d}](\mathtt{c},\mathtt{d})).$$

Simple matching problems enjoy unicity of solutions when resolved by application of the simple-matching algorithm as proved in Theorem 5.4.11. However, in order to mechanise elementary rewriting, one must also provide a notion of matching capable of dealing with equivariant formulæ, as in (Cheney, 2010). The next example shows how the property of meta-level equivariance affects the rewrite relation.

Example 6.2.7.

- Rewrite rule $R = (\mathtt{a} \rightarrow \mathtt{a})$ induces rewrite $\vdash \mathtt{b} \xrightarrow{\mathtt{R}} \mathtt{b}$ for each $\mathtt{b} \in \mathscr{A}$. This is due to equivariance and the application of meta-permutation $(\mathtt{a}\ \mathtt{b})$.
- If $\mathtt{R}' = \mathtt{X} \rightarrow [\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X}$ is a rewrite rule then, $\vdash \mathtt{Y} \xrightarrow{\mathtt{R}'} [\mathtt{b} \mapsto \mathtt{a}]\cdot\mathtt{Y}$ and $\vdash \mathtt{Y} \xrightarrow{\mathtt{R}'} [\mathtt{c} \mapsto \mathtt{d}]\cdot\mathtt{Y}$ and also $\vdash \mathtt{Y} \xrightarrow{\mathtt{R}'} [\mathtt{a} \mapsto \mathtt{c}]\cdot\mathtt{Y}$ by means of meta-permutations $(\mathtt{a}\ \mathtt{b})$, $(\mathtt{a}\ \mathtt{c})(\mathtt{b}\ \mathtt{d})$ and $(\mathtt{b}\ \mathtt{c})$ respectively.
- If $\mathtt{R}'' = \mathtt{X} \rightarrow \mathtt{X}$ is a rewrite rule then, $\vdash [\mathtt{a} \mapsto [\mathtt{b}]\mathtt{b}]\cdot\mathtt{Y} \xrightarrow{\mathtt{R}''} [\mathtt{a} \mapsto [\mathtt{b}]\mathtt{b}]\cdot\mathtt{Y}$ and also $\mathtt{b}\#\mathtt{Y} \vdash [\mathtt{a} \mapsto [\mathtt{b}]\mathtt{b}]\cdot\mathtt{Y} \xrightarrow{\mathtt{R}''} [\mathtt{b} \mapsto [\mathtt{c}]\mathtt{c}]\widehat{}(\mathtt{b}\ \mathtt{a})\cdot\mathtt{Y}$. This is because $\mathtt{b}\#\mathtt{Y} \vdash [\mathtt{a} \mapsto [\mathtt{b}]\mathtt{b}]\cdot\mathtt{Y} \approx_\alpha [\mathtt{b} \mapsto [\mathtt{c}]\mathtt{c}]\widehat{}(\mathtt{b}\ \mathtt{a})\cdot\mathtt{Y}$ as shown in Definition 6.2.4.

The definition of rewrite relation given in Definition 6.2.4 takes into account equivariant rules by means of using a meta-permutation π in the description of the one-step rewrite relation. As a result, the equivariant closure of the set of rules does not need to be included in the definition of a rewrite system as in (Fernández and Gabbay, 2007). This fact is important and therefore we highlight it by providing the following remark.

Remark 6.2.8. When rewriting with a rewrite system $\mathscr{R} = (\Sigma, Rw)$ we assume that, for any rewrite rule $R \in Rw$, if $R^\pi \in Rw$ then it can only be that $\pi = \mathtt{Id}$, that is, no permuted variants of rewrite rules are allowed in a rewrite system.

However, the property of equivariance in a rewrite system has the same effect as the use of meta-permutation in the one-step rewrite relation. We prove this claim in Lemma 6.2.11 Intuitively, this is the case since the rewrite relation is closed under permutations. Additionally,

in first-order rewrite systems, the one-step rewrite relation satisfies the properties of closure under context application and closure under substitution (Dershowitz and Jouannaud, 1990). This is also the case for extended nominal rewriting. In Theorem 6.2.10, we demonstrate that the properties of closure under context application (Claim 1), closure under substitution (Claim 2) and closure under permutations (Claim 3) are indeed satisfied. This is an extension of Theorem 50 in (Fernández and Gabbay, 2007).

The following technical lemma is applied on the proof of Theorem 6.2.10.

Lemma 6.2.9. *If* $\nabla \vdash s|_p \approx_\alpha t$ *then* $\nabla \vdash s[t]_p \approx_\alpha s$.

Proof. By induction on the derivation of $\nabla \vdash s|_p \approx_\alpha t$ and Definition 2.1.13. We omit the proof. ■

Theorem 6.2.10. *Let p be a position, π a permutation and θ a v-substitution. Suppose that $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ using rule $R = (\nabla \vdash l \to r)$. Then,*

1. *For any pair of nominal terms u, v, if $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ and $\Delta \vdash u[t]_{p'} \approx_\alpha v$ then $\Delta \vdash u[s]_{p'} \to_{\langle R,p,\pi,\theta\rangle} v$ (closure under context application);*

2. *Let $\Gamma \vdash \Delta\sigma$ where Γ is consistent. Then, it is the case that $\Gamma \vdash s\sigma \to_{\langle R,p,\pi,\theta\rangle} t\sigma$ (closure under substitution);*

3. *For any permutation π', if $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ then $\Delta \vdash \pi' \cdot s \to_{\langle R,p,(\pi' \circ \pi),(\pi' \cdot \theta)\rangle} \pi' \cdot t$ (closure under permutation).*

Proof. See Appendix C. ■

Now we are able to demonstrate that equivariance closure applied to a rewrite system is equivalent to the use of π in the one-step rewrite relation.

Lemma 6.2.11. *If* $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ *then* $s \to_{\langle R^\pi,p,\mathtt{Id},\theta\rangle} t$.

Proof. See Appendix C. ■

The definition of confluence and *critical pair* for nominal rewriting in the presence of a-substitutions extends naturally from standard NRSs and thus omitted here. Additionally, our unification algorithm plays a key role on computing critical pairs for extended rewrite rules. We direct the reader to (Fernández and Gabbay, 2007) for a study of confluence in the non-extended rewriting framework and to (Ayala-Rincón et al., 2016) for a more in-depth discussion, new results and further examples.

Elementary rewriting is of theoretical and practical interest. In (Bengtson and Parrow, 2009) an axiomatisation of the π-calculus is provided by means of a standard nominal syntax.

Further, in (Fernández and Gabbay, 2005) nominal rewriting is extended with a quantifier for modelling name generation, applied to a reaction system modelling an asynchronous nominal π-calculus. In the former, explicit substitution is handled by a nominal function whereas in the latter it is defined by means of reaction rules. In Example 6.2.5 we provide a representation of a rule in the π-calculus where explicit substitution was handled syntactically. Therefore, we argue that an axiomatisation of π-calculus could benefit from our extended nominal syntax and the machinery provided by elementary rewriting to induce rewrite steps on terms-in-context containing unabstracted atoms.

In the next section, we extend the operational characterisation of closedness by distinguishing a class of closed rules for which the generation of rewrites can be automated by the simple-matching algorithm given in Definition 5.4.6. Roughly, we refer to *closed-simple rewrite rules*, that is, simple rules without unabstracted atoms and where rewriting cannot change the binding status of an atom. For this class of closed rewrite rules the simple-matching algorithm is sufficient to provide a unique principal solution to the problem of matching a rule pattern to a term (see Theorem 5.4.11) without having to deal with equivariance.

6.3 Closedness of Extended Terms

In (Domínguez and Fernández, 2014) we present a translation from ground nominal terms and closed nominal rules to CRS terms and rules. Closedness of nominal terms and nominal rules is decidable in NRSs by application of the nominal matching algorithm and the use of freshened variants (Fernández et al., 2004). Similarly, closedness of extended nominal terms is also decidable by the extended matching algorithm defined in Definition 5.2.6. However, we are interested on checking closedness for the class of simple rewrite rules and terms since it is for this class that one can generate deterministic one-step rewrite steps with respect to a closed-simple rule when closed rewriting, as we soon show. As a result, the operational characterisation of closedness is extended from standard to simple extended terms. This is done next.

6.3.1 Closed-simple terms and rules

In the previous chapter, we introduced simple matching constraints (see Definition 5.4.1) as a set of matching constraints that enjoys the property of unique principal solutions when solved by the simple-matching algorithm (see Theorem 5.4.11). Additionally, closed terms and closed rules are not affected by meta-level equivariance since unabstracted atoms are

not allowed. In fact, permuted variants of closed terms are α-equivalent because of the property of object-level equivariance stated on the second claim of Corollary 3.1.5. Then, the set of extended rewrite rules which are both closed and simple has a special interest since meta-permutations are trivial and pattern matching enjoys the property of unique most general unifier.

To provide a notion of closedness in the extended formalism we need to have a suitable notion of *freshened variant*. A **freshened variant** of an extended term (resp. rewrite rule) is a variant of such term (resp. rewrite rule) where all atom and variable symbols are new. Here are some examples.

Example 6.3.1.

- The extended nominal term $\mathtt{cons}([\mathtt{a}^{\nu} \mapsto \mathtt{H}^{\nu}]\cdot\mathtt{F}^{\nu}, \mathtt{map}([\mathtt{a}^{\nu}]\mathtt{F}^{\nu}, \mathtt{T}^{\nu}))$ is a freshened variant of $\mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{T}))$,
- the primitive constraint $\mathtt{a}^{\nu}\#\mathtt{X}^{\nu}$ is a freshened variant of $a\#X$, and
- the rewrite rule $\mathtt{par}(\mathtt{out}(\mathtt{a}^{\nu}, \mathtt{b}^{\nu}), \mathtt{in}(\mathtt{a}^{\nu}, [\mathtt{c}^{\nu}]\mathtt{P}^{\nu})) \to \mathtt{par}([\mathtt{c}^{\nu} \mapsto \mathtt{b}^{\nu}]\cdot\mathtt{P}^{\nu}, \mathtt{Q}^{\nu})$ is a freshened variant of $\mathtt{par}(\mathtt{out}(\mathtt{a}, \mathtt{b}), \mathtt{in}(\mathtt{a}, [\mathtt{c}]\mathtt{P})) \to \mathtt{par}([\mathtt{c} \mapsto \mathtt{b}]\cdot\mathtt{P}, \mathtt{Q})$.

The definition of closed-simple terms and rules follows.

Definition 6.3.2 (Closedness of simple terms and rules)**.** A simple term-in-context $\nabla \vdash t$ is **closed-simple** when

$$(\nabla^{\nu} \vdash t^{\nu}) \overset{s}{?}\approx (\nabla, A(\nabla^{\nu}, t^{\nu})\#V(\nabla, t) \vdash t)$$

has a solution σ.

Call $R = (\nabla \vdash l \to r)$ **closed-simple** when $\nabla \vdash (l, r)$ is closed-simple.

Following Remark 5.4.9, if the matching constraint is simple, then the constraint is checked by the three-phase simple-matching algorithm given in Definition 5.4.8. Accordingly, if a solution exists to the simple matching constraint, such solution is principal, as stated in Theorem 5.4.11. Rule patterns must be simple in order for the matching algorithm to produce unique principal solutions whereas terms on the RHS of rules are restricted solely to being closed. However, although of theoretical interest, in practice rewrite systems do not contain rules with capture-avoidance substitutions on left patterns. If such substitutions occur, they do so in right rules. The rewrite systems from Example 6.2.2 and Example 6.2.3 belong to this class of closed-simple rules where a-substitutions occur only on RHS, as it also does all the systems that arise from functional programming. In these cases rules are always simple when represented as a tuple yet one still needs Definition 6.3.2 when checking whether they are also closed. Using the simple-matching algorithm in these cases is more

efficient than checking closedness with the matching algorithm since it may generate more candidate solutions during the matching process.

Examples of closed-simple terms and rules follow.

Example 6.3.3 (Closed-simple terms)**.**

- The term-in-context $\vdash \mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{H}))$ is closed-simple since solution $[\mathtt{H}^{\prime\prime} \mapsto \mathtt{H}; \mathtt{F}^{\prime\prime} \mapsto (\mathtt{a}^{\prime\prime}\ \mathtt{a})\cdot\mathtt{F}]$ solves the simple matching problem

  $$(\vdash \mathtt{cons}([\mathtt{a}^{\prime\prime} \mapsto \mathtt{H}^{\prime\prime}]\cdot\mathtt{F}^{\prime\prime}, \mathtt{map}([\mathtt{a}^{\prime\prime}]\mathtt{F}^{\prime\prime}, \mathtt{H}^{\prime\prime}))) \overset{s}{?}\approx (\mathtt{a}^{\prime\prime}\#\mathtt{F}, \mathtt{a}^{\prime\prime}\#\mathtt{H} \vdash \mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{H}))).$$

- The term-in-context $\vdash \mathtt{a}\#\mathtt{X} \vdash \mathtt{fun}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X}, [\mathtt{a} \mapsto \mathtt{X}]\cdot\mathtt{Y})$ is not closed-simple: the matching problem it generates is not simple, there is no fixed occurrence with trivial a-substitution of variable $\mathtt{X}$ nor of variable $\mathtt{Y}$. Further, variable $\mathtt{b}$ occurs unabstracted in the term;

- The term-in-context $[\mathtt{a} \mapsto [\mathtt{b}]\mathtt{b}]\hat{}(\mathtt{c}\ \mathtt{d})\cdot\mathtt{X}$ is not closed-simple, but it is closed. Similarly to the definition of closedness in NRSs (Fernández and Gabbay, 2007), *linear* terms without unabstracted atoms are closed.

Example 6.3.4 (Closed-simple rules)**.**

- Rule $\mathtt{fun}([\mathtt{a} \mapsto \mathtt{b}]\cdot\mathtt{X}, [\mathtt{a}]\mathtt{X}) \to [\mathtt{a}]\mathtt{X}$ is not closed-simple since subterm b is not closed.

- Rule $\mathtt{fun}([\mathtt{a}]\mathtt{X}, \mathtt{Y}) \to [\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}$ is closed-simple since the rule pattern is simple and solution $[\mathtt{X}^{\prime\prime} \mapsto (\mathtt{a}\ \mathtt{a}')\cdot\mathtt{X}; \mathtt{Y}^{\prime\prime} \mapsto \mathtt{Y}]$ solves the simple pattern matching problem

  $$(\vdash (\mathtt{fun}([\mathtt{a}^{\prime\prime}]\mathtt{X}^{\prime\prime}, \mathtt{Y}^{\prime\prime}), [\mathtt{a}^{\prime\prime} \mapsto \mathtt{Y}^{\prime\prime}]\cdot\mathtt{X}^{\prime\prime})) \overset{s}{?}\approx (\mathtt{a}^{\prime\prime}\#\mathtt{X}, \mathtt{a}^{\prime\prime}\#\mathtt{Y} \vdash (\mathtt{fun}([\mathtt{a}]\mathtt{X}, \mathtt{Y}), [\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X})).$$

- Rule $\mathtt{b}\#\mathtt{X} \vdash \mathtt{g}([\mathtt{b}](\mathtt{a}\ [\mathtt{b}]\mathtt{Y})\cdot\mathtt{X}, [\mathtt{a}]\mathtt{X}, [\mathtt{b}]\mathtt{Y}) \to [\mathtt{b}](\mathtt{a}\ \mathtt{b})\cdot\mathtt{X}$ is also closed-simple since the left rule pattern is simple and solution $[\mathtt{X}^{\prime\prime} \mapsto (\mathtt{a}^{\prime\prime}\ \mathtt{a})\cdot\mathtt{X}; \mathtt{Y}^{\prime\prime} \mapsto (\mathtt{b}^{\prime\prime}\ \mathtt{b})\cdot\mathtt{Y}]$ solves the simple pattern matching problem

  $(\mathtt{b}^{\prime\prime}\#\mathtt{X}^{\prime\prime} \vdash (\mathtt{g}([\mathtt{b}^{\prime\prime}][\mathtt{a}^{\prime\prime} \mapsto [\mathtt{b}^{\prime\prime}]\mathtt{Y}^{\prime\prime}]\cdot\mathtt{X}^{\prime\prime}, [\mathtt{a}^{\prime\prime}]\mathtt{X}^{\prime\prime}, [\mathtt{b}^{\prime\prime}]\mathtt{Y}^{\prime\prime}), [\mathtt{b}^{\prime\prime}](\mathtt{a}^{\prime\prime}\ \mathtt{b}^{\prime\prime})\cdot\mathtt{X}^{\prime\prime})) \overset{s}{?}\approx$
  $(\mathtt{b}\#\mathtt{X}, \mathtt{b}^{\prime\prime}\#\mathtt{X}, \mathtt{b}^{\prime\prime}\#\mathtt{Y}, \mathtt{a}^{\prime\prime}\#\mathtt{X}, \mathtt{a}^{\prime\prime}\#\mathtt{Y} \vdash (\mathtt{g}([\mathtt{b}][\mathtt{a} \mapsto [\mathtt{b}]\mathtt{Y}]\cdot\mathtt{X}, [\mathtt{a}]\mathtt{X}, [\mathtt{b}]\mathtt{Y}), [\mathtt{b}](\mathtt{a}\ \mathtt{b})\cdot\mathtt{X})).$

The following lemma will be used in Theorem 6.4.4.

Lemma 6.3.5. *$\Delta \vdash t$ is closed-simple when t is simple and there exists a solution σ with $\mathscr{D}om(\sigma) \subseteq V(\Delta^{\prime\prime} \vdash t^{\prime\prime})$ such that $\nabla, A(\Delta^{\prime\prime} \vdash t^{\prime\prime})\#V(\Delta \vdash t) \vdash (\Delta^{\prime\prime}\sigma,\ t \approx_\alpha t^{\prime\prime}\sigma)$.*

Proof. It follows directly from Definition 6.3.2 and Definition 5.4.5 for a pattern matching solution. ■

Closed-simple rules are well-behaved since rewriting does not generate fresh atoms or variables. Also, equivariance is trivially satisfied. Next, we extend closed rewriting for our formalism.

6.4 Closed Rewriting

In (Fernández et al., 2004), closed rewriting was introduced as an efficiently computable rewrite relation with respect to equivariant rewriting (which it is NP-complete (Cheney, 2004a)), obtained by avoiding equivariant matching when inducing rewrite steps. The conditions to achieve such result stemmed from the fact that rewrite rules are insensitive to the particular choice of names occurring in it. Then, by using atom and variable symbols which are fresh for both the pattern and the matched term, we can assume that meta-permutations are trivial and find closed rewrites by application of the standard nominal unification algorithm. The standard unification algorithm is complete for pattern matching during the generation of closed rewrites (Fernández and Gabbay, 2010). This property also applies to our general matching algorithm as shown in Theorem 5.3.18. However, it is for the set of closed-simple rules that the closed rewrite relation is deterministic. Therefore, we begin the section by extending the definition of closed rewriting to extended nominal syntax and then prove some properties about closed rewriting with closed-simple rules. The definition below follows the notation given for closed rewriting of non-extended terms in (Fernández and Gabbay, 2010).

Definition 6.4.1 (Closed rewriting)**.**

- A rewrite rule $R = (\nabla \vdash l \to r)$ induces a **(one-step) closed rewriting** $\Delta \vdash s \xrightarrow{R}_c t$ on terms s,t as follows: $\exists(\nabla^\prime \vdash l^\prime \to r^\prime) \in R^\prime$, that is, $R^\prime$ is a freshened variant of R (so fresh for R and $\Delta \vdash (s,t)$), $p \in \mathscr{P}os(s)$ and v-substitution θ such that

$$\frac{\Delta, A(R^\prime)\#V(\Delta,s,t) \vdash (\nabla^\prime\theta, \quad s|_p \approx_\alpha l^\prime\theta, \quad s[r^\prime\theta]_p \approx_\alpha t)}{\Delta \vdash s \xrightarrow{R}_c t} \; (\to\mathbf{Rew_c})$$

- The **(multi-step) closed rewrite relation** $\Delta \vdash_R s \to_c t$ is the reflexive transitive closure of the one-step closed rewrite, similarly as in Definition 6.2.4 for the multi-step rewrite relation.

So, the definition of closed rewriting corresponds to the definition of closed rewriting in the standard nominal formalism as given in (Fernández and Gabbay, 2010). However, there

is an implicit freshness context Δ^{ϕ} fresh for $R^{\prime}$ required for the application of a-substitutions to abstraction terms occurring after an instantiation of some variable in the rule.

Closed rewriting is preferred to extended elementary rewriting because by working with freshened variants one can derive some information about freshness of matched terms with respect to rule patterns. Take for instance pattern $a\#X \vdash X \to X$ and term-in-context $\vdash Z$, there is always some atom $a \in \mathscr{A}$ such that $a\#Z$ however, if closed rewriting is not used, matching is unsuccessful because of $\nvdash a\#Z$.

As in standard nominal rewriting, it is possible to do elementary rewriting with a closed rule, closed rewriting with a (non-closed) rule or closed rewriting with a closed rule. Further, closed rewriting with a closed rule generates the same rewrites as elementary rewriting with the same closed rule and a meta-permutation. This is true for NRSs and also for our extended formalism *when working with the set of closed-simple rules*. We prove this claim after a few clarifying examples.

Example 6.4.2. The term-in-context $t = \vdash \mathtt{par}(\mathtt{out}(\mathtt{a},\mathtt{b}),\mathtt{in}(\mathtt{a},[\mathtt{c}]\mathtt{X}))$ representing a π-calculus term with signature from Example 2.1.1 is in normal form when closed rewriting with the π-calculus rewrite rule representation from Example 6.2.5.

A freshened variant of the rule and the term-in-context is

$$\mathtt{R}^{\prime} = (\mathtt{a}^{\prime}\#\mathtt{X}, \mathtt{b}^{\prime}\#\mathtt{X}, \mathtt{c}^{\prime}\#\mathtt{X} \vdash \mathtt{par}(\mathtt{out}(\mathtt{a}^{\prime},\mathtt{b}^{\prime}),\mathtt{in}(\mathtt{a}^{\prime},[\mathtt{c}^{\prime}]\mathtt{P}^{\prime})) \to [\mathtt{c}^{\prime} \mapsto \mathtt{b}^{\prime}]\cdot\mathtt{P}^{\prime})$$

therefore the matching problem is unsolvable because there is no v-substitution σ that makes $a \approx_{\alpha} a^{\prime}\sigma$ and $b \approx_{\alpha} b^{\prime}\sigma$.

Example 6.4.3. The term-in-context $\vdash \mathtt{map}([\mathtt{a}]\mathtt{sub}(\mathtt{a}),\mathtt{cons}(\mathtt{0},\mathtt{nil}))$ closed rewrites to a normal form using the rewrite system from Example 6.2.2 (denoted below by $\mathscr{R}$) and the signature for summation introduced in Example 2.1.1 as follows.

$\vdash_{\mathscr{R}} \mathtt{map}([\mathtt{a}]\mathtt{suc}(\mathtt{a}),\mathtt{cons}(\mathtt{0},\mathtt{nil})) \to_{c} \mathtt{cons}(\mathtt{suc}(\mathtt{0}),\mathtt{map}([\mathtt{a}^{\prime}]\mathtt{suc}(\mathtt{a}^{\prime},\mathtt{nil})))$
$\to_{c} \mathtt{cons}(\mathtt{suc}(\mathtt{0}),\mathtt{nil})$.

A classic proof for NRSs is that elementary rewriting with closed rules is equivalent to closed rewriting. We show next the property also holds for closed-simple rules in eNRSs.

Theorem 6.4.4. *Given a term-in-context $\Delta \vdash s$, if $R = (\nabla \vdash l \to r)$ is closed-simple then $\Delta \vdash s \xrightarrow{R} t$ implies $\Delta \vdash s \xrightarrow{R}_{c} t$.*

Proof. The result follows by the fact that pattern matching closed-simple rules enjoys unique principal solutions (see Theorem 5.4.11), Lemma 6.3.5 and Proposition 5.15 in (Fernández and Gabbay, 2010) where it was shown the property holds for standard closed rewriting. ■

6.5 Conclusion

We have provided a general definition of extended nominal rewrite rule, named as *simple rule*, and described two versions of nominal rewriting: elementary rewriting, which must deal with equivariant matching, and closed rewriting, where atoms in the term do not deal explicitly with atoms in the rewrite rule since rules are closed-simple and the rewrite relation follows an approach similar to the Barendregt variable convention. The simple-matching algorithm provided in Chapter 5 can then be used to mechanise the rewriting process when restrictions to simple rewrite rules discard the explicit handling of equivariance.

We are now ready to provide an encoding of eNRSs to both NRSs and CRSs and also an encoding of CRSs to eNRSs. This is done in the next chapter.

Chapter 7

From eNRSs to CRSs and Back Again

This chapter aims to provide an encoding of eNRSs into CRSs and vice versa.

The chapter begins by translating a class of eNRSs into CRSs. Roughly, this class contains eNRS rules which are closed-simple and where a-substitutions only occur on RHS terms. A more generalised translation, allowing a-substitutions on rule patterns is possible yet non-trivial. We have decided to postpone its definition as future work and focus on a translation of eNRSs rules which are similar in structure to CRS rules. The translation has an intermediate step where eNRSs is reduced to NRSs with explicit substitution, using term-former `sub` and its behaviour as given in the set of rules in Definition 1.5.4. Then, NRSs to CRSs has already been proven reduction-preserving and such Theorem is used to conclude the claim that eNRS can be encoded in CRSs. A more direct translation is possible, that is, translating directly eNRSs to CRSs without the use of term-former `sub` since CRSs uses λ-calculus as part of its substitution calculus. However, one needs a structural definition of closedness for extended terms to show the preservation of closedness when providing a direct translation.

The chapter concludes defining a translation of CRSs to eNRSs. This translation is straight-forward since one just needs to transform explicit a-substitutions into implicit ones and update the already given proofs.

7.1 From eNRS to CRS Systems

In this section we provide a transformation of a class of eNRS to CRS systems by means of an intermediate step where eNRS rules are converted into NRS rules. Extended rewrite rules belonging to this class are closed-simple, have a term-former as root symbol on the pattern term and a-substitutions are only allowed on RHS terms. This transformation shows the relation of such class of eNRSs and NRSs and it is then used to relate eNRSs and CRSs.

CRSs already have a notion of higher-order substitution and does not need to explicitly define it, however, to provide the required proofs of preservation of closedness one must be able to use a structural definition of closed extended terms as the one given by (Clouston, 2007) for non-extended terms. This is left for future work and instead we make substitution explicit on CRSs by making use of term-former `sub` when reducing eNRSs to NRSs. Then, the set of explicit substitution rules given in Definition 1.5.4 is added to any translated CRS system and a-substitutions in extended rules are transformed into function applications using term-former `sub` from such definition. Finally, we conclude the section by showing that the rewrite step is preserved between both translations, namely, from eNRS to NRS and from NRS to CRS systems.

7.1.1 Converting extended to standard nominal terms

Prior reduction of extended to non-extended nominal terms, one must transform suspended simultaneous a-substitutions into sequential applications so that the semantics are preserved when simulating a-substitution behaviour by means of the explicit substitution rules. However, the notation of a-substitutions cannot be modified to sequential binding applications, therefore some renaming of atoms needs to take place to simulate sequential a-substitutions with simultaneous mappings. This is done next.

Renaming a-substitutions

.

The auxiliary function below will be used on the translation of extended to standard nominal terms given in Definition 7.1.5.

Definition 7.1.1. Let t be an extended nominal term and Δ^{ϕ}, Γ a pair of freshness contexts. Then, $\langle \varnothing, t \rangle = (\Delta^{\phi}, t')$ where $\langle \Gamma, s \rangle$ is a function inductively defined over extended nominal terms s such that, for the case where $(s = \phi\hat{}\pi \cdot X)$ we have,

- $\langle \Gamma, \phi\hat{}\pi \cdot X \rangle = (\Delta^{\phi}, \phi'\hat{}(\pi' \circ \pi) \cdot X)$ where
- $\phi = [a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]$,
- Let $a'_1, \ldots, a'_n$ be new names such that $\Gamma_i \vdash a'_j \# s_i, \pi \cdot X$ for $1 \leq i, j \leq n$. Then,
- $\phi' = [a'_1 \mapsto s'_1; \ldots; a'_n \mapsto s'_n]$ with $(\Gamma'_i, s'_i) = \langle \Gamma_i \cup \Gamma, s_i \rangle$ for $1 \leq i \leq n$ and
- $\Delta^{\phi} = \bigcup_{1 \leq i \leq n} \Gamma'_i$ and also
- $\pi' = (a_1\ a'_1) \cdots (a_n\ a'_n)$.

And for the case $(s = (s_1, \ldots, s_n))$ we have,

- $\langle \Gamma, (s_1, \ldots, s_n) \rangle = (\Delta^{\phi}, (s'_1, \ldots, s'_n))$ where
- $(\Gamma_i, s'_i) = \langle \Gamma, s_i \rangle$ for $i \leq i \leq n$ and
- $\Delta^{\phi} = \bigcup_{1 \leq i \leq n} \Gamma_i$.

The rest of the inductive cases are trivial thus we omit them.

Informally, function $\langle \cdot, \cdot \rangle$ is applied to an extended nominal term s to transform each variable occurrence of form $\phi \hat{} \pi \cdot X$ in s into an occurrence $\phi' \hat{} (\pi' \circ \pi) \cdot X$ by replacing $\mathscr{D}om(\phi)$ with a set of fresh names of the same cardinality as $\mathscr{D}om(\phi)$ and adding a permutation $(a\ a') \in \pi'$ to permutation π for each bijection $(a\ a')$ between the original atom a in $\mathscr{D}om(\phi)$ and its new atom replacement a' in the modified a-substitution ϕ'. New names a' used to rename the domain of a-substitutions are locally generated and preserved as primitive constraints in $\{a' \# Y \mid Y \in V(\mathscr{I}mg(\phi))\}$ generated by application of the freshness relation over the set of terms which are below the position of the variable symbol X, guaranteeing that no name clashes will occur during an instantiation of the variable.

The parsing of a sequence of a-substitutions is done all at once. The domain of a-substitutions is replaced by the set of new atoms and the image of a-substitutions is taken as a tuple and used as argument to function $\langle \cdot, \cdot \rangle$. Recall that we fixed the positioning to a lexicographic order, thus the replacement of the domain of a-substitution to new atoms may lead to changes in the representation of a-substitutions depending on the order of the new atoms. However, the change in the representation does not affect the result since new atoms have been chosen to avoid interference during sequential application of the a-substitution. Accordingly, in the related proofs we will not take the parsing order into consideration.

An example follows.

Example 7.1.2. The term $\mathtt{t} = \mathtt{f}([\mathtt{a} \mapsto (\mathtt{b}, \mathtt{c}); \mathtt{b} \mapsto (\mathtt{a}, \mathtt{c}); \mathtt{c} \mapsto (\mathtt{a}, \mathtt{b})] \cdot \mathtt{X})$ is transformed into the term-in-context $\{a' \# X, b' \# X, c' \# X\} \vdash t'$ as follows

$$\langle \varnothing, \mathtt{t} \rangle = (\{\mathtt{a}' \# \mathtt{X}, \mathtt{b}' \# \mathtt{X}, \mathtt{c}' \# \mathtt{X}\}, \mathtt{t}' = \mathtt{f}([\mathtt{a}' \mapsto (\mathtt{b}, \mathtt{c}); \mathtt{b}' \mapsto (\mathtt{a}, \mathtt{c}); \mathtt{c}' \mapsto (\mathtt{a}, \mathtt{b})] \hat{} (\mathtt{a}\ \mathtt{a}')(\mathtt{b}\ \mathtt{b}')(\mathtt{c}\ \mathtt{c}') \cdot \mathtt{X}))$$

according to Definition 7.1.1.

The lemma below shows that the properties of the nominal term are preserved by the translation. This is the case because of the syntax-directed translation function and Remark 2.1.14, where we stated that syntax trees differing only in the positioning of a-substitutions suspended over the same variable occurrence are α-equivalent.

Lemma 7.1.3. *Given t and t' such that $(\Delta^{\phi}, t') = \langle \varnothing, t \rangle$ by Definition 7.1.1. Then, $\Delta^{\phi} \vdash t \approx_{\alpha} t'$.*

Proof. By induction on the structure of t, Remark 2.1.14 and the one-to-one correspondence between the elements of t and t' as shown in the syntax-directed translation in Definition 7.1.1.

The interesting case is that of the variable, the rest of the cases are trivial and thus omitted. Following Definition 7.1.1 we have $\langle \Gamma, [a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]\hat{}\pi \cdot X \rangle =$
$(\Delta^{\phi}, [a'_1 \mapsto s'_1; \ldots; a'_n \mapsto s'_n]\hat{}((a_1\ a'_1)\ldots(a_n\ a'_n) \circ \pi) \cdot X)$ where $(\Gamma'_i, s'_i) = \langle \Gamma \cup \Gamma_i, s_i \rangle$ and $\Gamma_i \vdash a'_j \# s'_i$ by the definition, for $1 \leq i, j \leq n$. Then, $\Delta^{\phi} = \bigcup_{1 \leq i \leq n} \Gamma'_i$ such that, by application of rule $(\approx_{\alpha}\mathbf{X})$, $\Delta^{\phi} \vdash [a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]\hat{}\pi \cdot X \approx_{\alpha} [a'_1 \mapsto s'_1; \ldots; a'_n \mapsto s'_n]\hat{}(((a_1\ a'_1)\ldots(a_n\ a'_n)) \circ \pi) \cdot X$ where $\Delta^{\phi} \vdash a'_i \# X$ and $\Delta^{\phi} \vdash s_i \approx_{\alpha} s'_i$ by inductive hypothesis for $1 \leq i \leq n$. The property holds. ■

Example 7.1.4. Applying v-substitution $\theta(\mathtt{X}) = \mathtt{g}([\mathtt{a}]\mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{c})$ to term $\mathtt{t}$ and term $\mathtt{t}'$ (obtained from $(\Delta^{\phi} = \{\mathtt{a}'\#\mathtt{X}, \mathtt{b}'\#\mathtt{X}, \mathtt{c}'\#\mathtt{X}\}, \mathtt{t}')$) from Example 7.1.2 we observe that
$\mathtt{t}\theta = \mathtt{f}(\mathtt{g}([\mathtt{a}]\mathtt{a}, (\mathtt{b}, \mathtt{c}), (\mathtt{a}, \mathtt{c}), (\mathtt{a}, \mathtt{b})))$
$\mathtt{t}'\theta = \mathtt{f}(\mathtt{g}([\mathtt{a}']\mathtt{a}', (\mathtt{b}, \mathtt{c}), (\mathtt{a}, \mathtt{c}), (\mathtt{a}, \mathtt{b})))$ where $\vdash \Delta^{\phi}\theta$
and $\vdash \mathtt{f}(\mathtt{g}([\mathtt{a}]\mathtt{a}, (\mathtt{b}, \mathtt{c}), (\mathtt{a}, \mathtt{c}), (\mathtt{a}, \mathtt{b}))) \approx_{\alpha} \mathtt{f}(\mathtt{g}([\mathtt{a}']\mathtt{a}', (\mathtt{b}, \mathtt{c}), (\mathtt{a}, \mathtt{c}), (\mathtt{a}, \mathtt{b})))$

7.1.2 Reducing extended to non-extended nominal terms

Definition 7.1.5 (From extended to non-extended nominal terms)**.** Let t be an extended nominal term, $\mathtt{sub}$ a binary term-former denoting higher-order substitution as given by the set of rewrite rules in Definition 1.5.4. Then, $\mathtt{toNom}(t) = (\Delta^{\phi}, \mathtt{toSub}(t')) = (\Delta^{\phi}, \bar{t})$ where $(\Delta^{\phi}, t') = \langle \varnothing, t \rangle$ as in Definition 7.1.1 and $\bar{t} = \mathtt{toSub}(t')$ where $\mathtt{toSub}$ is inductively defined over the structure of extended nominal terms as follows

(atom)	$\mathtt{toSub}(a)$	$=$	a.
(abs)	$\mathtt{toSub}([a]s')$	$=$	$[a]\mathtt{toSub}(s')$.
(fun)	$\mathtt{toSub}(fs')$	$=$	$f\mathtt{toSub}(s')$.
(tuple)	$\mathtt{toSub}(s_1, \ldots, s_n)$	$=$	$(\mathtt{toSub}(s_1), \ldots, \mathtt{toSub}(s_n))$
(var)	$\mathtt{toSub}(\phi\hat{}\pi \cdot X)$	$=$	$\mathtt{sub}([a_n] \cdots \mathtt{sub}([a_1]\pi \cdot X, \mathtt{toSub}(s_1)), \mathtt{toSub}(s_n))$

where $\phi = [a_1 \mapsto s_1; \ldots a_n \mapsto s_n]$ or $\pi \cdot X$ if $\phi = \mathtt{Id}$

Informally, the definition above states that a translation of an extended term t to a non-extended one is a term-in-context $\Delta^{\phi} \vdash \bar{t}$, from the resulting pair $(\Delta^{\phi}, \bar{t})$, where the encoding is syntax-directed for all cases except the variable case where, if the a-substitution is trivial, the translation is also trivial. Otherwise, implicit a-substitution $[a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]$ suspended over some variable $\pi \cdot X$ is encoded as an explicit a-substitution
$\mathtt{sub}([a_n] \cdots \mathtt{sub}([a_1]\pi \cdot X, s_1) \cdots, s_n)$ where application of each function application $\mathtt{sub}([a_i]\text{-}, s_i)$ is done in a lexicographic order with respect to the name of the abstractor $[a_i]$- for $1 \leq i \leq n$. However, although we have fixed such an order, any other order will do because there is no

name clashes when applying the a-substitutions sequentially since Definition 7.1.1 has been applied to term t prior the translation to non-extended term.

A clarifying example follows.

Example 7.1.6. Given extended nominal term t and pair (Δ^{ϕ}, t') from Example 7.1.2, the translation of t to Definition 7.1.5 is as follows

$$\texttt{toNom(t)} = (\Delta^{\phi}, \overline{\texttt{t}} = \texttt{f(sub([c']sub([b']sub([a'](a a')(b b')(c c')}\cdot\texttt{X,(b,c)),(a,c)),(a,b)).}$$

Property 7.1.7 (Preservation of variable and atom occurrences)**.** *Suppose t is an extended nominal term and $(\Delta^{\phi}, \bar{t}) = \texttt{toNom}(t)$ its translation according to Definition 7.1.5. For each occurrence of variable $X \in \mathscr{X}$ (resp. atom $a \in \mathscr{A}$) in t there is a corresponding occurrence of X (resp. a) in $\bar{t}$. Therefore no atom or variable occurrences are added during the translation of t.*

Proof. The translation is syntax-directed for all cases but the variable case where for each $\phi\hat{}\pi\cdot X$ in t where $\phi = [a_1 \mapsto s_1; \ldots; a_n \mapsto s_n]$ there is $\texttt{sub}([a'_n]\cdots\texttt{sub}([a'_1]\pi\cdot X, \texttt{toSub}(s'_1))\cdots, \texttt{toSub}(s'_n)) = \texttt{toSub}(\phi'\hat{}(\pi' \circ \pi)\cdot X)$ in $\bar{t}$ where the occurrence of symbol X is preserved and the generated abstractors $[a_i]$- do not add occurrences of a_i to $\bar{t}$ for $1 \leq i \leq n$ and $\phi' = [a'_1 \mapsto s'_1; \ldots; a'_n \mapsto s'_n]$, $\pi' = (a_1\ a'_1)\ldots(a_n\ a'_n)$ by Definition 7.1.1. The result follows by inductive hypothesis on $s'_1, \ldots, s'_n$. ■

The next property shows that closedness is preserved when transforming extended nominal terms into non-extended ones. This will be useful when proving closedness preservation for the CRS rules produced by our extended rule translation in Definition 7.1.13.

Property 7.1.8. *If $\Delta \vdash t$ is a closed extended nominal term-in-context then $\Delta, \Delta^{\phi} \vdash \bar{t}$ is also closed where $(\Delta^{\phi}, \bar{t}) = \texttt{toNom}(t)$ according to Definition 7.1.5.*

Proof. Suppose $\Delta \vdash t$ is closed, then there are no unabstracted atoms in t. Moreover, by Lemma 7.1.3 $\Delta, \Delta^{\phi} \vdash t \approx_{\alpha} t'$ and then $\Delta, \Delta^{\phi} \vdash t'$ is also closed where $(\Delta^{\phi}, t') = \langle \varnothing, t \rangle$ by Definition 7.1.1. Now, $\texttt{toSub}(t') = \bar{t}$ by Definition 7.1.5 and by Property 7.1.7 the term-in-context $\Delta, \Delta^{\phi} \vdash \bar{t}$ does not have additional variable or atom occurrences. The result follows from the fact that $\Delta, \Delta^{\phi} \vdash t'$ is closed and therefore if $[a \mapsto s]\cdot v$ occurs as a (closed) subterm in t' then, by Definition 7.1.5, $\texttt{sub}([a]v, s)$ occurs in $\bar{t}$ where terms s and v are also closed in t' since $\Delta^{\phi} \vdash a\#v, s$ by Definition 7.1.1. Note that for the case where $\Delta \vdash t$ is a closed extended nominal term-in-context with `Id` a-substitutions, then $\bar{t} = t$. ■

7.1.3 Converting extended nominal rules

Definition 7.1.9 (Simple rule translation)**.** The translation of a simple rewrite rule $R = (\nabla \vdash l \to r)$ is $(\nabla \cup \Delta_l^\phi \cup \Delta_r^\phi, \bar{l}, \bar{r}) = \mathtt{toNRS}(\nabla, l, r)$ where $\mathtt{toNom}(l) = (\Delta_l^\phi, \bar{l})$ and $\mathtt{toNom}(r) = (\Delta_r^\phi, \bar{r})$ according to Definition 7.1.5.

The following examples illustrate the above definition.

Example 7.1.10. Rewrite rule R_π from Example 6.1.2 is translated according to Definition 7.1.9 as follows:

$$\mathtt{a'\#P} \vdash \mathtt{par(out(a,b),in(a,[c]P))} \to \mathtt{sub([a']}(\mathtt{a'\ c})\cdot\mathtt{P,b)}$$

Rewrite rules in CRSs are closed and therefore extended nominal rules must also be closed in order to be translated between formalisms.

Example 7.1.11. The translation of the rules in Example 6.2.2 according to Definition 7.1.9 is as follows:

$$\begin{array}{rlcl} & \mathtt{map([a]F,nil)} & \to & \mathtt{nil} \\ \mathtt{b\#H,b\#F} \vdash & \mathtt{map([a]F,cons(H,T))} & \to & \mathtt{cons(sub([b](a\ b)\cdot F,H),map([a]F,T)).} \end{array}$$

Example 7.1.12. The higher-order function `foldl` with signature $\mathtt{foldl} : 3, \mathtt{nil} : 0, \mathtt{cons} : 2$ could be described as the following eNRSs:

$$\begin{array}{ll} (\mathbf{foldl_{nil}}) & \mathtt{foldl([a][b]F,Z,nil)} \to \mathtt{Z} \\ (\mathbf{foldl_{cons}}) & \mathtt{foldl([a][b]F,Z,cons(H,T))} \to \mathtt{foldl([a][b]F,[a \mapsto Z; b \mapsto H]\cdot F,T).} \end{array}$$

Its translation according to Definition 7.1.9 is as follows:

$$\begin{array}{lll} (\mathbf{foldl'_{nil}}) & & \mathtt{foldl([a][b]F,Z,nil)} \to \mathtt{Z} \\ (\mathbf{foldl'_{cons}}) & \mathtt{c\#\{F,H,Z\},d\#\{F,H,Z\}} \vdash & \mathtt{foldl([a][b]F,Z,cons(H,T))} \to \\ & \mathtt{foldl([a][b]F,sub([d]sub([c](a\ c)(b\ d)\cdot F,Z),H),T).} & \end{array}$$

Below, we impose some conditions on eNRS rules to obtain a class of rules that can be translated to CRSs.

Definition 7.1.13 (Standard extended nominal rule)**.** An extended simple nominal rewrite rule is called **standard-simple** when it is closed-simple, a-substitutions occur uniquely in the right-hand side term of the simple rule and the left-hand side has the form fs for any $f \in \Sigma$.

Now we have standard-simple rules and standard rules. The former follows the definition above and the latter follows the definition given in Definition 1.4.3. Additionally, and to avoid further confusion, nominal terms as described in (Urban et al., 2004) are described as non-extended, that is, we omit referring to them as standard nominal terms.

Lemma 7.1.14 (Well-defined rule translation)**.** *Let $R = (\nabla \vdash l \to r)$ be a standard-simple rule. If $\overline{R} = (\nabla, \Delta^{\phi} \vdash \overline{l} \to \overline{r})$ is its translation according to Definition 7.1.9 then $\overline{R}$ is a standard nominal rule as defined in Definition 1.4.3.*

Proof. The definition of standard-simple for extended rules subsumes the definition of standard for non-extended rules (see Definition 1.4.3) therefore one just needs to show that closedness is preserved during the translation to a NRS rule. The result follows directly by application of Property 7.1.8 to the term-in-context $\nabla, \Delta^{\phi} \vdash (\overline{l}, \overline{r})$. ■

7.1.4 Preservation of reduction between translations

We are now ready to show that the encoding of eNRSs into CRSs preserves the rewrite relation. To do so, we show first that the rewrite relation is preserved when reducing eNRSs to NRSs and then we are able to use Theorem 1.4.7 from the NRSs to CRSs translation to prove the rest.

Theorem 7.1.15 (Preservation of reduction in NRSs)**.** *Let $R = (\nabla \vdash l \to r)$ be a standard extended nominal rule. Let u be a ground nominal term. If $u \to_{\langle R,p,\pi,\theta\rangle} v$ then $u \to^{+}_{\langle \overline{R} \cup \mathscr{R}_{\sigma},p,\pi,\theta\rangle} v$ using $\overline{R} = \texttt{toNRS}(\nabla, l, r)$ as given in Definition 7.1.9 and the explicit substitution rules, $\mathscr{R}_{\sigma}$, from Definition 1.5.4.*

Proof. Suppose $u \to_{\langle R,p,\pi,\theta\rangle} v$ where we assume without loss of generality that $\pi = \texttt{Id}$ since R is a closed rule. By Definition 6.2.4 we are able to derive $\vdash \nabla\theta$, $\vdash u|_p \approx_{\alpha} l\theta$, $\vdash u[r\theta]_p \approx_{\alpha} v$ where $\vdash r\theta \approx_{\alpha} t$. Now, $\overline{R} = (\Delta \vdash \overline{l} \to \overline{r})$ is a standard-simple rule (see Definition 7.1.13) by Lemma 7.1.14 and, trivially, $\vdash u|_p \approx_{\alpha} \overline{l}\theta$ since a-substitutions are not allowed on the pattern and $\vdash l\theta \approx_{\alpha} \overline{l}\theta$ by Lemma 2.5.12. By Theorem 2.3.6 we have $\vdash nf_{\sigma}(\overline{r}\theta) \approx_{\alpha} t$ where nf_{σ} is the function described in Definition 1.5.11 which applies the explicit substitution rules $\mathscr{R}_{\sigma}$. Hence, it is the case that $\vdash_{R \cup \mathscr{R}_{\sigma}} u[\overline{r}\theta] \to v$. The result follows by Definition 6.2.4. ■

The preservation of reduction is stated for ground terms only since CRS terms are ground. However, there exists also a reduction-preserving relation between implicit and explicit substitution for non-ground terms where variables have suspended a-substitutions. This is left for future work.

Example 7.1.16. The ground (closed) nominal term $\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{suc}(0),\mathtt{cons}(\mathtt{suc}(0),\mathtt{nil}))$ rewrites to a normal form in two steps using the eNRSs from Example 7.1.12 and the signature for *summation* from Example 2.1.1 as follows:

$$\begin{aligned}
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{suc}(0),\mathtt{cons}(\mathtt{suc}(0),\mathtt{nil}))\\
&\quad\to\langle(\mathbf{foldl_{cons}}),\varepsilon,\mathtt{Id},[\mathtt{F}\mapsto\mathtt{add}(\mathtt{a},\mathtt{b});\mathtt{Z}\mapsto\mathtt{suc}(0);\mathtt{H}\mapsto\mathtt{suc}(0);\mathtt{T}\mapsto\mathtt{nil}]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{add}(\mathtt{suc}(0),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\mathbf{foldl_{nil}}),\varepsilon,\mathtt{Id},[\mathtt{F}\mapsto\mathtt{add}(\mathtt{a},\mathtt{b});\mathtt{Z}\mapsto\mathtt{add}(\mathtt{suc}(0),\mathtt{suc}(0))]\rangle\\
&\mathtt{add}(\mathtt{suc}(0),\mathtt{suc}(0)).
\end{aligned}$$

The same ground term rewrites to a normal form in 10 steps using the NRSs from Example 7.1.12, the rules from explicit substitution from Definition 1.5.4 and the signature for *summation* from Example 2.1.1 as follows:

$$\begin{aligned}
&\mathtt{c}\#\{\mathtt{F},\mathtt{H},\mathtt{Z}\},\mathtt{d}\#\{\mathtt{F},\mathtt{H},\mathtt{Z}\}\vdash\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{suc}(0),\mathtt{cons}(\mathtt{suc}(0),\mathtt{nil}))\\
&\quad\to\langle(\mathbf{foldl'_{cons}}),\varepsilon,\mathtt{Id},[\mathtt{F}\mapsto\mathtt{add}(\mathtt{a},\mathtt{b});\mathtt{Z}\mapsto\mathtt{suc}(0);\mathtt{H}\mapsto\mathtt{suc}(0);\mathtt{T}\mapsto\mathtt{nil}]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{sub}([\mathtt{d}]\mathtt{sub}([\mathtt{c}]\mathtt{add}(\mathtt{c},\mathtt{d}),\mathtt{suc}(0)),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\mathbf{f}}),1{\cdot}2{\cdot}1{\cdot}1{\cdot}1,\mathtt{Id},[\mathtt{X}\mapsto\mathtt{add}(\mathtt{a},\mathtt{d});\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{sub}([\mathtt{d}]\mathtt{add}\ \mathtt{sub}([\mathtt{a}](\mathtt{a},\mathtt{d}),\mathtt{suc}(0)),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\mathbf{prod}}),1{\cdot}2{\cdot}1{\cdot}1{\cdot}1{\cdot}1,\mathtt{Id},[\mathtt{X}\mapsto(\mathtt{a},\mathtt{d});\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{sub}([\mathtt{d}]\mathtt{add}(\mathtt{sub}([\mathtt{a}]\mathtt{a},\mathtt{suc}(0)),\mathtt{sub}([\mathtt{a}]\mathtt{d},\mathtt{suc}(0))),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\mathbf{var}}),1{\cdot}2{\cdot}1{\cdot}1{\cdot}1{\cdot}1{\cdot}1,\mathtt{Id},[\mathtt{X}\mapsto\mathtt{a};\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{sub}([\mathtt{d}]\mathtt{add}(\mathtt{suc}(0),\mathtt{sub}([\mathtt{a}]\mathtt{d},\mathtt{suc}(0))),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\varepsilon}),1{\cdot}2{\cdot}1{\cdot}1{\cdot}1{\cdot}1{\cdot}2,\mathtt{Id},[\mathtt{X}\mapsto\mathtt{d};\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{sub}([\mathtt{d}]\mathtt{add}(\mathtt{suc}(0),\mathtt{d}),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\mathbf{f}}),1{\cdot}2,\mathtt{Id},[\mathtt{X}\mapsto\mathtt{add}(\mathtt{suc}(0),\mathtt{a});\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{add}\ \mathtt{sub}([\mathtt{a}](\mathtt{suc}(0),\mathtt{a}),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\mathbf{prod}}),1{\cdot}2{\cdot}1,\mathtt{Id},[\mathtt{X}\mapsto(\mathtt{suc}(0),\mathtt{a});\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{add}(\mathtt{sub}([\mathtt{a}]\mathtt{suc}(0),\mathtt{suc}(0)),\mathtt{sub}([\mathtt{a}]\mathtt{a},\mathtt{suc}(0))),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\varepsilon}),1{\cdot}2{\cdot}1{\cdot}1,\mathtt{Id},[\mathtt{X}\mapsto\mathtt{suc}(0);\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{add}(\mathtt{suc}(0),\mathtt{sub}([\mathtt{a}]\mathtt{a},\mathtt{suc}(0))),\mathtt{nil})\\
&\quad\to\langle(\sigma_{\varepsilon}),1{\cdot}2{\cdot}1{\cdot}2,\mathtt{Id},[\mathtt{X}\mapsto\mathtt{a};\mathtt{Y}\mapsto\mathtt{suc}(0)]\rangle\\
&\mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{add}(\mathtt{a},\mathtt{b}),\mathtt{add}(\mathtt{suc}(0),\mathtt{suc}(0)),\mathtt{nil})\\
&\quad\to\langle(\mathbf{foldl_{nil}}),\varepsilon,\mathtt{Id},[\mathtt{F}\mapsto\mathtt{add}(\mathtt{a},\mathtt{b});\mathtt{Z}\mapsto\mathtt{add}(\mathtt{suc}(0),\mathtt{suc}(0))]\rangle\\
&\mathtt{add}(\mathtt{suc}(0),\mathtt{suc}(0)).
\end{aligned}$$

Below, we show that the translation from eNRS to CRS systems preserves the rewrite relation.

Theorem 7.1.17 (Preservation of reduction in CRSs)**.** *Let $R = (\nabla \vdash l \to r)$ be a standard extended nominal rule. Let u be a ground nominal term and therefore a CRS term by Remark 1.5.6. If $u \to_{\langle R,p,\pi,\theta\rangle} v$ then $u \Rightarrow^{+}_{\hat{R}\cup\hat{\mathscr{R}}_\sigma} \hat{v}$ using $\hat{v} = \mathscr{T}(\varnothing, v)$ and $\overline{R} = \texttt{toNRS}(\nabla, l, r)$ as given in Definition 7.1.9 and also the explicit substitution rules, $\mathscr{R}_\sigma$, from Definition 1.5.4 such that for each rule $R_\sigma \in \mathscr{R}_\sigma$, there is rule $\hat{R}_\sigma \in \mathscr{R}_\sigma$ by means of Definition 1.4.4 for the translation of NRS rules to CRS rules.*

Proof. Suppose $u \to_{\langle R,p,\pi,\theta\rangle} v$. Then, by Lemma 7.1.15 we obtain $u \to^{+}_{\langle \overline{R}\cup\mathscr{R}_\sigma,p,\pi,\theta\rangle} v$ where $\overline{R}$ is a standard nominal rule. The result follows by Theorem 1.4.7 for the preservation of reduction from NRS to CRS systems applied to each step in the rewrite relation $u \to^{+}_{\langle \overline{R}\cup\mathscr{R}_\sigma,p,\pi,\theta\rangle} v$. ■

The corollary below follows directly from its analogue in standard NRSs (see Corollary 1.4.8).

Corollary 7.1.18 (Termination)**.** *Termination of the translated CRS implies termination of the eNRS.*

7.2 From CRS to eNRS Systems

In this section, we propose a more efficient approach to the translation of CRSs to the nominal rewriting formalism by using the syntax of nominal terms extended with a-substitutions from Chapter 2. The translation builds on top of the previous translation algorithm defined in (Domínguez and Fernández, 2015). As a result, definitions and claims previously stated in (Domínguez and Fernández, 2015) for the translation of CRSs to NRSs must be revised and updated in order to reflect the modifications to the syntax of nominal terms. However, updating proofs and claims for the CRS to NRSs translation algorithm is straight-forward because of Lemma 2.3.5 where it was shown that the action of a-substitutions on ground terms corresponds to the notion of higher-order substitution in CRSs. The syntax for extended nominal terms along with the definition of closed rewriting in the presence of a-substitutions (see Definition 6.4.1) offers an improved approach for the translation algorithm by reducing the number of steps to reach a normal form; this is achieved by shifting the action of atom substitutions to the meta-level. Then, it is no longer required to extend a given NRS with the set of rules for explicit substitution.

7.2.1 Converting CRS meta-terms

In a CRS rule, meta-applications of form $Z_i^n t$ occurring on the LHS meta-term of such rule can only contain variables in t. Substitution of a CRS variable by another is handled in the nominal formalism by means of a swapping. Then, the left function translation $Left(\cdot)$ given in 1.5.3 is sufficient to translate CRSs to eNRSs. This is not the case for the right function translation where any syntactical structure is allowed in the tuple t in $Z_i^n t$. Accordingly, we begin the section by extending the translation function for meta-terms on the RHS of CRS rules.

Definition 7.2.1 (Translation of right meta-terms)**.** Let $l \Rightarrow r$ be a CRS rule. Let Φ_l be the function defined in Definition 1.5.1 applied to the CRS meta-term l. Then $Right_e(l,r) = (\Delta_r, \langle\!\langle r \rangle\!\rangle_{\Phi_l})$ where

$$\Delta_r = \{a_k \# Z_i \mid Z_i^n \in MV(r), a_k \text{ occurs bound above } Z_i^n\}$$

and $\langle\!\langle r \rangle\!\rangle_{\Phi_l}$ is defined by:

$$\begin{array}{lcl}
\langle\!\langle a \rangle\!\rangle_{\Phi_l} & = & a \\
\langle\!\langle f s \rangle\!\rangle_{\Phi_l} & = & f \langle\!\langle s \rangle\!\rangle_{\Phi_l} \\
\langle\!\langle [a]s \rangle\!\rangle_{\Phi_l} & = & [a] \langle\!\langle s \rangle\!\rangle_{\Phi_l} \\
\langle\!\langle (t_1,\ldots,t_n) \rangle\!\rangle_{\Phi_l} & = & (\langle\!\langle t_1 \rangle\!\rangle_{\Phi_l},\ldots,\langle\!\langle t_n \rangle\!\rangle_{\Phi_l}) \\
\langle\!\langle Z_i^n t \rangle\!\rangle_{\Phi_l} & = & \left\{ \begin{array}{ll} Z_i & \text{, if } n = 0; \\ \phi \hat{} \pi \cdot Z_i & \text{, otherwise (where } t = (t_1,\ldots,t_n) \text{ such that} \end{array} \right. \\
 & & \phi = [\Phi_l(Z_i^n)_{m1} \mapsto \langle\!\langle t_{m1} \rangle\!\rangle_{\Phi_l}; \ldots; \Phi_l(Z_i^n)_{mk} \mapsto \langle\!\langle t_{mk} \rangle\!\rangle_{\Phi_l}] \\
 & & \pi = (\Phi_l(Z_i^n)_{j1}\ \langle\!\langle t_{j1} \rangle\!\rangle_{\Phi_l}) \cdot \ldots \cdot (\Phi_l(Z_i^n)_{jk}\ \langle\!\langle t_{jk} \rangle\!\rangle_{\Phi_l}), \text{ and} \\
 & & t_{j1},\ldots,t_{jk},t_{m1},\ldots,t_{mk} \in \{t_1,\ldots,t_n\} \text{ where} \\
 & & \langle\!\langle t_{j1} \rangle\!\rangle_{\Phi_l},\ldots,\langle\!\langle t_{jk} \rangle\!\rangle_{\Phi_l} \in \mathscr{A}
\end{array}$$

The main distinction with the original translation definition in 1.5.5 is the translation of the meta-application. In the translated term, substitution of atoms by terms is now represented implicitly, suspended over the variable symbol, instead of the explicit substitution notation using term-former `sub`.

Proving that the RHS translating function for extended terms returns closed terms-in-context is reduced to the proof for its non-extended analogue because of the correspondence between the extended nominal syntax and the rules for explicit substitution used in the previous version. This correspondence was proved in Lemma 2.3.5 and the claim about the translation preserving closedness is shown below, after the revision of two auxiliary lemmas.

Lemma 7.2.2 (Preservation of CRS meta-variables as NRS variables in the $Right_e$ translation)**.** *Let Φ_l be the function defined in Definition 1.5.1 applied to the CRS meta-term l. Suppose t is a CRS meta-term and $(\Delta, t') = Right_e(l,t)$ its translation by Definition 7.2.1. Then, $Z_i^n(t_1,\ldots,t_n) \lhd t$ if and only if $\phi \hat{} \pi \cdot Z_i \lhd t'$.*

Proof. By induction over the structure of t. The proof follows from the proof given in (Domínguez and Fernández, 2015, Lemma 6.11). ■

Lemma 7.2.3 (Preservation of free variables as unabstracted atoms in the $Right_e$ translation)**.** *Let l,r be a pair of closed CRS meta-terms, Φ_l as defined in Definition 1.5.1 and $(\Delta, r') = Right_e(l,r)$ as in Definition 7.2.1. Assume $«s»_{\Phi_l} = s'$ is computed in the translation of r, where s is any subterm of r (e.g. $s = r$). Then, variable a is free in s if and only if a is an unabstracted atom in s'. Hence, there are no unabstracted atom subterms in r', since r is closed.*

Proof. By induction on the definition of $Right_e$. The proof is solved similarly to that in (Domínguez and Fernández, 2015, Lemma 6.12) and thus omitted. ■

Next, Lemma 6.14 in (Domínguez and Fernández, 2015) is updated to show that Definition 7.2.1 preserves closedness when translating CRS meta-terms on RHS of CRS rules. This is the case since closed CRS meta-terms do not have free variables, our translation is syntax-directed thus it does not introduce new atoms and respects the arity of meta-variables. Additionally, it was shown in Theorem 2.3.6 that application of a-substitutions to ground terms is equivalent to using the set of explicit substitution rules $\mathtt{sub}([a]t,s)$ and function nf_σ from Definition 1.5.4 and Definition 1.5.11 respectively.

Lemma 7.2.4. *Let t be the RHS meta-term of a CRS rule following Barendregt's naming convention, Φ the function defined in Definition 1.5.1 such that for each meta-variable Z_i^n in t, $Z_i^n \in dom(\Phi_s)$ for the LHS meta-term s such that $\Phi_s(Z_i^n) = [a_1,\ldots,a_n]$.*
Let $Right_e(s,t) = (\Delta_t, «t»_{\Phi_s})$ be the translation as in Definition 7.2.1, then $\Delta_t \vdash «t»_{\Phi_s}$ is a closed term-in-context.

Proof. By induction on the syntax of t. It follows by (Domínguez and Fernández, 2015, Lemma 6.14) and Theorem 2.3.6. ■

7.2.2 Converting CRS rules

Next, the definition of CRS rule translation is updated to incorporate the translation function given in Definition 7.2.1.

Definition 7.2.5. We define the translation of the CRS rule $l \Rightarrow r$ as $\mathscr{C}_e^{\mathscr{R}}(l,r) = \Delta \vdash l' \to r'$, where $Left(l) = (\Delta_l, l')$, $Right_e(l,r) = (\Delta_r, r')$ and $\Delta = \Delta_l \cup \Delta_r$.

A few clarifying examples follow.

Example 7.2.6. The translation of the β-rule shown in (Domínguez and Fernández, 2015, Example 6.17.) is now translated, according to Definition 7.2.5, as

$$\vdash \mathtt{app}(\mathtt{lam}([\mathtt{a}]\mathtt{Z}), \mathtt{Z}') \to [\mathtt{a} \mapsto \mathtt{Z}']\cdot\mathtt{Z}.$$

Example 7.2.7 (Summation rules)**.** The summation rules with signature as in Example 2.1.1 could be described in CRSs by

$$\begin{array}{lcl} \Sigma([\mathtt{a}]\mathtt{F}(\mathtt{a}), 0) & \Rightarrow & \mathtt{F}(0); \\ \Sigma([\mathtt{a}]\mathtt{F}(\mathtt{a}), \mathtt{suc}(\mathtt{N})) & \Rightarrow & \mathtt{g}(\Sigma([\mathtt{b}]\mathtt{F}(\mathtt{b}), \mathtt{N}), \mathtt{F}(\mathtt{suc}(\mathtt{N}))). \end{array}$$

The translation of the rules by means of Definition 7.2.5 is

$$\begin{array}{lcl} \vdash \Sigma([\mathtt{a}]\mathtt{F}, 0) & \to & [\mathtt{a} \mapsto 0]\cdot\mathtt{F}; \\ \mathtt{b}\#\mathtt{F} \vdash \Sigma([\mathtt{a}]\mathtt{F}, \mathtt{suc}(\mathtt{N})) & \to & \mathtt{g}(\Sigma([\mathtt{b}](\mathtt{a}\ \mathtt{b})\cdot\mathtt{F}, \mathtt{N}), [\mathtt{a} \mapsto \mathtt{suc}(\mathtt{N})]\cdot\mathtt{F}) \end{array}$$

Example 7.2.8 (Map and Foldl rules)**.** The Haskell functions `map` and `foldl` have the following definition in CRSs

$$\begin{array}{lcl} \mathtt{map}([\mathtt{a}]\mathtt{F}(\mathtt{a}), \mathtt{nil}) & \Rightarrow & \mathtt{nil}; \\ \mathtt{map}([\mathtt{a}]\mathtt{F}(\mathtt{a}), \mathtt{cons}(\mathtt{H}, \mathtt{T})) & \Rightarrow & \mathtt{cons}(\mathtt{F}(\mathtt{H}), \mathtt{map}([\mathtt{b}]\mathtt{F}, \mathtt{T})) \\ \\ \mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{F}(\mathtt{a}, \mathtt{b}), \mathtt{N}, \mathtt{nil}) & \Rightarrow & \mathtt{N}; \\ \mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{F}(\mathtt{a}, \mathtt{b}), \mathtt{N}, \mathtt{cons}(\mathtt{H}, \mathtt{T})) & \Rightarrow & \mathtt{foldl}([\mathtt{c}][\mathtt{d}]\mathtt{F}(\mathtt{c}, \mathtt{d}), \mathtt{F}(\mathtt{N}, \mathtt{H}), \mathtt{T}) \end{array}$$

where function symbols have the same signature as in Example 6.2.2 and Example 7.1.12 respectively.

The translation of the rules by means of Definition 7.2.5 is

$$\begin{array}{rcl} \vdash \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{nil}) & \to & \mathtt{nil}; \\ \mathtt{b}\#\mathtt{F} \vdash \mathtt{map}([\mathtt{a}]\mathtt{F}, \mathtt{cons}(\mathtt{H}, \mathtt{T})) & \to & \mathtt{cons}([\mathtt{a} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{map}([\mathtt{b}](\mathtt{a}\ \mathtt{b})\cdot\mathtt{F}, \mathtt{T})) \\ \\ \vdash \mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{F}, \mathtt{N}, \mathtt{nil}) & \to & \mathtt{N}; \\ \mathtt{c}\#\mathtt{F}, \mathtt{d}\#\mathtt{F} \vdash \mathtt{foldl}([\mathtt{a}][\mathtt{b}]\mathtt{F}, \mathtt{N}, \mathtt{cons}(\mathtt{H}, \mathtt{T})) & \to & \\ \mathtt{foldl}([\mathtt{c}][\mathtt{d}](\mathtt{a}\ \mathtt{c})(\mathtt{b}\ \mathtt{d})\cdot\mathtt{F}, [\mathtt{a} \mapsto \mathtt{N}; \mathtt{b} \mapsto \mathtt{H}]\cdot\mathtt{F}, \mathtt{T}) & & \end{array}$$

7.2.3 Preservation of the rewrite relation in eNRSs

Below, Lemma 7.2.9 asserts preservation of closedness when translating CRS rules to eNRS rules. Then, Lemma 7.2.10 shows that translating first a pair of a RHS meta-term and a valuation and producing the instance of the result is equivalent modulo alpha to first applying the valuation to the meta-term and then translating the instance term. These two lemmas will be of support later, when stating Theorem 7.2.11.

Lemma 7.2.9. *Let $R = l \Rightarrow r$ be a CRS rule. If $\Delta \vdash l' \to r'$ is its translation according to Definition 7.2.5, then $\Delta \vdash l' \to r'$ is a standard-simple nominal rule.*

Proof. The proof is equivalent to that stated in (Domínguez and Fernández, 2014, Lemma 6.18.), using Lemma 7.2.4 for the RHS term of $\Rightarrow$. ■

In the next lemma we update the correctness property for the RHS instantiation.

Lemma 7.2.10 ($Right_e$ instantiation)**.** *Let $l \Rightarrow r$ be a CRS rule and Φ_l the function given in Definition 1.5.1 applied to l. Assume $Right_e(l,r) = \Delta \vdash r'$ where, according to Definition 7.2.1, $\Delta = \{a_k \# Z_i^n \mid a_k$ occurs bound above Z_i^n in $r\}$. Let σ be a safe valuation such that $dom(\sigma) \subseteq MV(l)$ (hence, $\sigma(r)$ is a CRS term).*

Suppose $s' = \text{«}s\text{»}_{\Phi_l}$ is a recursive call in the translation of $Right_e(l,r)$, for any subterm s of r (e.g., $s = r$), and $\sigma' = \langle\sigma\rangle_{\phi_l}$ by Definition 1.5.8.

Then, $\text{«}\sigma(s)\text{»}_{\Phi_\varnothing} \approx_\alpha s'\sigma'$ and σ' satisfies Δ, i.e., $\vdash \Delta\sigma'$.

Proof. By induction on the structure of s, using the fact that $\nabla \vdash r'$ is closed by Lemma 7.2.4, the syntactic equivalence between CRS terms and ground nominal terms given in Remark 1.5.6 and the safety conditions given in Definition 1.3.9 and Remark 1.3.8. The proof follows closely that in (Domínguez and Fernández, 2015, Lemma 6.24). We update the case for meta-variables.

- The case ($s = Z_i^n t$).
  - For the case where $t = ()$, the result follows similarly to case (1) of (Mvar) in (Domínguez and Fernández, 2015, Lemma 6.24).
  - For the case where $t = (t_1, \ldots, t_n)$, there is $\Phi_l(Z_i^n) = [a_1, \ldots, a_n]$ by Definition 1.5.1 where $[a_1, \ldots, a_n]$ is a list of distinct atoms by Remark 1.3.8 and then $\text{«}Z_i^n(t_1, \ldots, t_n)\text{»}_{\Phi_l} = [a_{m1} \mapsto \text{«}t_{m1}\text{»}_{\Phi_l}; \ldots; a_{mk} \mapsto \text{«}t_{mk}\text{»}_{\Phi_l}]\widehat{\ }(a_{j1}\ \text{«}t_{j1}\text{»}_{\Phi_l}) \cdots (a_{jk}\ \text{«}t_{jk}\text{»}_{\Phi_l}) \cdot Z_i$ where $j1, \ldots jk, \ldots, m1, \ldots mk \in \{1, \ldots, n\}$ and $\text{«}t_{j1}\text{»}_{\Phi_l}, \ldots, \text{«}t_{jk}\text{»}_{\Phi_l} \in \mathscr{A}$ by Definition 7.2.1.

    Assume, without loss of generality, $\sigma(Z_i^n) = \underline{\lambda}(b_1 \ldots b_n).s_i$. Following Definition 1.5.8 for $\langle\sigma\rangle_{\Phi_l}$, $\sigma'(Z_i) = (a_1\ b_1) \cdots (a_n\ b_n) \cdot s_i$ where s_i is a nominal ground

term by Remark 1.5.6 and $\{a_1,\ldots,a_n\} \notin A(s_i)$ by Remark 1.3.8. Moreover, following the same remark we have $V(\sigma)\cap V(r)=\varnothing$, therefore it is the case that $\sigma'|_{Z_i}$ satisfies Δ.

Then, the instance of s' by σ' is defined as $s'\sigma' =$
$(((a_{j1}\ «t_{j1}»_{\Phi_l})\cdots(a_{jk}\ «t_{jk}»_{\Phi_l}))\cdot((a_1\ b_1)\cdots(a_n\ b_n)\cdot s_i))([a_{m1}\mapsto «t_{m1}»_{\Phi_l};\ldots;a_{mk}\mapsto «t_{mk}»_{\Phi_l}])$
$\approx_\alpha s_i\{b_1\mapsto «t_1»_{\Phi_l}\sigma'\}\cdots\{b_n\mapsto «t_n»_{\Phi_l}\sigma'\}$ by Lemma 2.3.5 where $\{\cdot\mapsto\cdot\}$ is the higher-order substitution of CRSs.

Now, $[\![\sigma(Z_i^n(t_1,\ldots,t_n))]\!]_{\Phi_\varnothing} = s_i\{b_1\mapsto[\![\sigma(t_1)]\!]_{\Phi_l}\}\cdots\{b_n\mapsto[\![\sigma(t_n)]\!]_{\Phi_l}\}$, and the result follows by induction hypothesis.

■

The next theorem is the main result of this section.

Theorem 7.2.11 (Translating CRS rewrite steps in the extended formalism)**.** *Let $R = l \Rightarrow r$ be a CRS rule. Let u be a CRS term.*
If $u \Rightarrow_R v$ then $\vdash u \xrightarrow{\mathbb{R}}_c v$ using $\mathbb{R} = \mathbb{C}_e^{\mathscr{R}}(l,r)$ and Definition 6.4.1 for the one-step closed rewriting.

Proof. By Definition 7.2.5, $\mathbb{R} = \mathbb{C}_e^{\mathscr{R}}(l,r) = \nabla_l\cup\nabla_r \vdash l' \to r'$ where $\mathbb{R}$ is a standard-simple rule by Lemma 7.2.9. Since u,v are terms in CRS, they are also ground terms in eNRS following Remark 1.5.6 and, without loss of generality, we can assume that $\nabla_l\cup\nabla_r\vdash l'\to r'$ does not mention any atom in u (i.e., it is already freshened for u).

Now, if $u\Rightarrow_R v$ then there exists a position p in u and a valuation σ where $MV(l) = dom(\sigma)$ such that $u|_p = \sigma(l)$ and $v = u[\sigma(r)]_p$.

Let $\sigma' = \langle\sigma\rangle_{\Phi_l}$ according to Definition 1.5.8. By Lemma 6.21 in (Domínguez and Fernández, 2015), if $Left(l) = (\nabla_l, l')$ then $Left(\sigma(l)) = (\varnothing, l'\sigma')$ where σ' satisfies ∇_l. Then, it is also the case that $\vdash u|_p \approx_\alpha l'\sigma'$ by Remark 1.5.6. Moreover, σ' also satisfies ∇_r since we are using Barendregt convention.

It remains to prove that $\vdash u[r'\sigma']_p \approx_\alpha v$. We have $Right_e(l,\sigma(r)) = (\varnothing, «\sigma(r)»_{\Phi_l})$ and by Lemma 7.2.10 $\vdash «\sigma(r)»_{\Phi_l} \approx_\alpha r'\sigma'$. Hence, $u\Rightarrow_R v$ with $v = u[\sigma(r)]_p$ implies $\vdash u\xrightarrow{\mathbb{R}}_c v$ where $v\approx_\alpha u[r'\sigma']_p$ and we are done. ■

7.3 Conclusion

We have provided a pair of reduction-preserving translations, one encoding a class of standard-simple eNRSs into CRSs and one encoding CRSs into eNRSs. The former is not necessarily direct due to the transformation of implicit a-substitutions into explicit ones. However, the

latter preserves and reflects the rewrite relation. Now we have a pair of translations which provide a direct encoding of CRSs into extended nominal rewriting and vice versa, that is, the translation from NRSs to CRSs given in (Domínguez and Fernández, 2014) and the translation from CRSs to eNRSs given in this chapter. Additionally, we have shown the relation between standard eNRSs and standard NRSs by defining a reduction-preserving translation from eNRSs to NRSs using term-former `sub` to represent explicit a-substitution.

A more generalised translation for extended rules where a-substitution is allowed on rule patterns is left for future work along with a direct translation of RHS terms to RHS meta-terms in rules preserving the implicit substitution on both formalisms, CRSs and eNRSs.

Part V

Conclusions

Chapter 8

Related Work

Comparing Higher-Order Rewriting formalisms Extensions of term rewrite systems with binding support, as well as translations among such extensions, have been the subject of a large amount of previous work, in particular between CRSs (Klop et al., 1993) and other generalisations of term rewrite systems. In this thesis, our comparison only considers CRSs versus (extended) NRSs however there are alternative studies comparing other higher-order rewriting formalisms. Below, we briefly comment on the most common higher-order formalisms and the studies comparing them with CRSs.

Omitting the λ-calculus, the first definition of a higher-order term rewriting system is that of Aczel's Contraction Schemes (Aczel, 1978) (CSs), where TRSs are extended with binders and meta-variables and therefore allowing higher-order functions to be defined. In CSs, abstractions are not considered terms on their own yet they can be used as subterms. The framework of CRSs is an extension on the ideas of Aczel's Contraction Schemes (Aczel, 1978). CSs can be seen as a restricted class of CRSs and in his thesis (Klop, 1980), Klop studies extensively the properties of CSs. Unlike CSs, in CRSs, as in most of the higher-order formalisms that came after, abstractions are considered as terms.

Next chronologically is Expression Reduction Systems (ERSs) by Khasidashvili (Khasidashvili, 1990) (later revised in (Glauert et al., 2005)), Higher-order Rewriting Systems by Nipkow (Nipkow, 1991) and Interaction systems of Asperti and Laneve (Asperti and Laneve, 1993), among others. In (van Oostrom and van Raamsdonk, 1994), it was shown that CRSs and HRSs have roughly the same expressive power and that their main difference is in the meta-language employed. Further, van Raamsdonk gives a description of all these systems in (van Raamsdonk, 1996) and shows the relations between them by means of Higher-Order Rewrite Systems, a formalism where all these distinct presentations can be translated.

CRSs and ERSs are close conceptually and their distinction is mainly syntactic, for instance, the restriction in ERSs to *admissible* assignments where the status of variables

is preserved after mapping variables to terms in rewrite rules. Similarly to both CRSs and eNRSs, there is a clear distinction in ERSs between variables and meta-variables. The difference with CRSs (or, the similarity with eNRSs) is that meta-variables have arity 0. Another similarity with eNRSs is that ERSs has an operator for *meta-substitution*, denoted by $(s_1/x_1, \ldots, s_n/x_n)t$ and where each s_i is an arbitrary meta-term and x_i has a binding effect in t, thus representing simultaneous substitution of variables by terms. However, meta-substitutions in ERSs deal implicitly with α-conversion when applied by working with an equivalence class of variables whereas α-conversion is elegantly handled in eNRSs by swappings and the freshness relation. More than one quantifier symbol can be used in ERSs to bind terms, arity of quantifiers is a pair of natural numbers where the first natural number expresses how many variable symbols the quantifier can bind whereas the second number states how many arguments the quantifier symbol is supposed to have. Take for instance operator λ from the λ-calculus, the λ operator is represented in ERSs by a quantifier of arity $(1, 1)$ as it binds just one variable and takes one argument. This feature enables ERSs to have a more natural representation but it is syntactic sugar in that ERSs can be expressed within a syntax having only one binder symbol (Glauert et al., 2005).

Nipkow's Higher-Order Rewriting Systems (Nipkow, 1991) (HRSs) and Jouannaud and Okada's Algebraic Functional Systems (Jouannaud and Okada, 1991) (AFSs) are the first higher-order formalisms which use types and were both introduced simultaneously at LICS 1991.

In HRSs, Nipkow suggested the simply-typed lambda calculus as a meta-language, generalising first-order terms to λ-terms in that terms are equivalence classes modulo $\alpha\beta\eta$. HRSs are an extension of both TRSs and λ-calculus introduced to investigate the meta theory of higher-order systems like Isabelle (Paulson, 1989, 1994) and λ-Prolog (Nadathur and Miller, 1988). Meta-terms appearing in HRSs rewrite rules are required to be in η-long normal form so a metavariable X representing a unary function has to be written as $\lambda x.Mx$. This can prove cumbersome when defining a rewriting system. The notion of complete development from CRSs is replaced in the typed HRSs by η-long β-normalisation, which it is more powerful. However, a simpler notion of substitution is adopted in eNRSs where meta-variables are substituted by a simple replacement and the notion of higher-order substitution has been axiomatised in the extended nominal syntax and thus dealt with explicitly in the metalanguage as we mentioned above. Pattern HRSs (Nipkow, 1991) are a subclass of HRSs where the left hand side of rewrite rules are patterns as defined by Miller (Miller, 1991a,b) (commonly known as higher-order patterns) so that unification (and thus matching) retains theoretical properties closer to first-order unification.

HRSs have a much different presentation than systems like CRSs, ERSs, CSs (and also eNRSs), as a result, the first formal comparison between formalisms was done by van Oostrom and van Raamsdonk in (van Oostrom and van Raamsdonk, 1994), designing reduction-preserving translations from one formalism to the other and vice versa. Translations from one system to the other are fairly simple because them both are based on the λ-calculus (van Oostrom and van Raamsdonk, 1994). Before moving to AFSs, we must mention Wolfram's Higher-Order Term Rewriting Systems (Wolfram, 1993) which are similar to HRSs. Higher-Order Term Rewriting Systems also have a simply typed λ-calculus as a metalanguage although rewrite rules in this system are more general than rewrite rules in HRSs.

Algebraic Functional Systems(Jouannaud and Okada, 1991) were introduced as a modelling of functional programming languages and extend the simply-typed λ-calculus with rewrite rules and function symbols. Then, the rewrite relation is generated by the union of the β reduction rule and the algebraic rewrite rule that induces the rewrite step (which may be higher-order). In AFSs there is no pattern restriction and matching is syntactic, so not modulo β. In (Kop, 2012), Kop designs a formalism close to AFS called Algebraic Functional Systems with Meta-variables (AFSM) to derive termination results for most of the common higher-order formalisms; one can find termination-preserving translations between AFSM and CSs, CRSs, AFSs, PRSs, among others.

The design concept of the Rewriting Calculus (Cirstea and Kirchner, 2001a,b), also called ρ-calculus, is to make explicit objects out of all the basic notions of rewriting such as rewrite rule formation, application and evaluation. The ρ-calculus combines the first-order and λ-calculus paradigms providing a general abstract mechanism that allows to abstract over a variable, for instance $\lambda x.t$ like in the λ-calculus or abstract over an elaborated pattern p, as in $\lambda p.t$. The latter is also written as $p \to t$ to emphasize the rewriting aspect of the construct. It also has an structure operator $(\_ \wr \_)$ and the (hidden) application operator. Then, the application of the first-order rule $x+0 \to x$ to the term $n+0$ is encoded in the ρ-calculus as $(\lambda \mathtt{plus}(x,0).x)plus(n,0)$ or the ρ-term $\lambda f(a).a \wr \lambda f(a)b$ represents the rewrite system consisting of the two rules $f(a) \to a$ and $f(a) \to b$. In (Bertolissi and Kirchner, 2007) systems of the rewriting calculus can be simulated by CRSs where ρ-terms are encoded as a set of CRS terms and as a set of CRS rules the encoding of the evaluation rules of the ρ-calculus. In (Bertolissi et al., 2006) the converse translation is addressed. In this case, the translation is more elaborated since matching is done implicitly in CRSs and therefore extra ρ-terms are needed to direct the reduction in the ρ-calculus.

In (Cheney, 2005), Cheney reduces higher-order pattern unification (Miller, 1991a,b) to nominal unification and in (Levy and Villaret, 2012), Levy and Villaret do the opposite, reducing nominal unification to higher-order pattern unification so that nominal unification

can be decided in quadratic time. Also in (Levy and Villaret, 2012), Levy and Villaret present a simplified extension by removing freshness equations and prove the correspondence between most general unifiers and most general pattern unifiers. Both (Cheney, 2005) and (Levy and Villaret, 2012) are closely related to our work on transforming CRSs to (e)NRSs and vice versa. However, whereas their interest is in preserving the unifiability relation, ours is in preserving the rewrite relation, a key concept in rewriting theories to derive properties like termination, confluence and completion. Another work close to ours (in the sense of relating nominal and higher-order formalisms) is (Gacek, 2010), Gacek's semantics-preserving translation from α-Prolog to a subclass of the programming logic found in Abella, a theorem proving system using a higher-order abstract syntax (Gacek, 2008).

On nominal unification The first to prove that nominal unification is decidable were Urban et al. in (Urban et al., 2004), providing a naïve algorithm which works by recursive descent and has exponential worst-case runtime due to not sharing subterms. Our matching algorithm for extended nominal terms in Chapter 5 is built on top of this one, however, we have looked at the properties of decidability, soundness and completeness and left out the time and space complexity for future work. Cheney proved that a more general form than nominal unification, called equivariant unification, is NP-complete (Cheney, 2010). Additionally, in the same paper, Cheney also defined a polynomial time algorithm for equivariant matching when the LHS does not have any swappings. Fernández and Calvès studied more efficient nominal unification algorithms based on optimisations taken from first-order unification implementations and managed to define and implement a polynomial time algorithm for nominal unification (Calvès and Fernández, 2007, 2008a,b) based on graph representation of terms and a lazy propagation of swappings. This result was later improved by two independent studies. Calvès and Fernández (Calvès, 2010; Calvès and Fernández, 2010), and Levy and Villaret (Levy and Villaret, 2010) independently presented a quadratic time nominal unification algorithm inspired on the Paterson-Wegman first-order unification algorithm (Paterson and Wegman, 1978). Levy and Villaret introduced the notion of replacement to handle nominal constraints and used multi-equations in a manner similar to Martelli and Montari's first-order unification algorithm (Martelli and Montanari, 1982), in order to deal with structural constraints. On the other hand, Calvès and Fernández used permutations and sets of atoms to handle nominal constraints and mimic the Paterson-Wegman algorithm when dealing with structural constraints. The two different approaches are later unified in (Calvès, 2013), resulting in a *general abstract nominal unification algorithm* that can be used to exchange properties between both approaches. Additionally, Kumar and Norrish have also studied efficient forms of nominal unification using *triangular*

substitutions which are not necessarily idempotent but are better suited for backtracking search in logic programming (α-prolog (Cheney and Urban, 2004)). Their results can be found in (Kumar and Norrish, 2010). Permissive nominal terms (Dowek et al., 2010) is a variation on the original nominal syntax to ease the writing of proofs. This is done by annotating variables with the atoms that can occur free, or not, inside and therefore discarding the freshness relation. In (Dowek et al., 2010), Dowek et al. also investigate the relation between permissive nominal unification and both nominal unification and higher-order pattern unification. Prior to (Levy and Villaret, 2010), Levy and Villaret already proved, in an indirect manner, that nominal unification can be decided in quadratic time. This was achieved by reducing nominal unification to higher-order pattern unification (Miller, 1991a,b) in (Levy and Villaret, 2012). Higher-order unification with explicit substitutions (Dowek et al., 1995, 2000) (and the case for higher-order patterns in (Dowek et al., 1996)) uses the $\lambda\sigma$-calculi presented in (Abadi et al., 1991) (a first-order rewriting system using de Bruijn notation which provides and explicit treatment of substitutions initiated by β-reductions) to reduce higher-order unification to equational unification with a theory that separates replacement and capture-avoidance substitution by distinguishing between unification variables and $\beta\eta$-conversion variables. Our matching algorithm for extended terms also makes such distinction between variables as it is one of the distinctive features of nominal terms, however, the interaction between atoms and variables is less complex due to swappings and the freshness relation.

The constraint handling proposed for our unitary matching algorithm in Chapter 5 was already introduced in the presentation of extended nominal terms given by Fairweather et al. in (Fairweather et al., 2015). However, our restriction is tighter than that in (Fairweather et al., 2015) since simple matching constraints add the condition that at least one variable occurrence with trivial substitution has to be *at a fixed position* (see Definition 5.4.1), whereas in (Fairweather et al., 2015) such occurrence could be either at a fixed or suspended position. We argue that this restriction is not enough to provide a unique principal solution, as they state in their results. Further, their approach may introduce fresh variables when used to generate the rewrite relation. Take for instance the matching constraint $([\mathtt{a} \mapsto \mathtt{Y}]\cdot\mathtt{X}, \mathtt{X}) \stackrel{?}{\approx} ([\mathtt{a} \mapsto \mathtt{M}]\cdot\mathtt{Z}, \mathtt{Z})$ and solutions $(\varnothing, \theta_1 = [X \mapsto Z][Y \mapsto M])$ and $(\{a\#Z\}, \theta_2 = [X \mapsto Z])$ to such constraint. Both solutions are principal ones because, although composition of θ_2 and $[Y \mapsto M]$ satisfies θ_1 in an empty freshness context, it is not the case that $\varnothing \vdash a\#Z[Y \mapsto M]$. Further, imagine $([a \mapsto Y]\cdot X, X)$ is the LHS of a rewrite rule where the RHS also contains variable Y, a rewrite step generated with θ_2 would then introduce a fresh variable in the rewrite relation.

The restriction to simple matching constraints and distinction with matching constraints postponed as equality constraints is *originally* proposed by Pfenning in the constraint simplification algorithm given in (Pfenning, 1991). Benefits following this approach are shared with Pfenning's technique, there is no backtracking or restriction on variable occurrences. However, our implementation of Pfenning's design is optional since we have shown that matching of extended terms is decidable (see Theorem 5.3.18). On the negative side, postponed constraints are not guaranteed to have a favourable solution. However, unlike Pfenning's design, unsolvable postponed constraints do not necessarily remain unknown until the end of the algorithm since we have added rules to deal with clashing equalities. Unification in Qu-Prolog (Nickolas and Robinson, 1996), a logic programming language, is related to unification of extended nominal terms. The term language allows for possibly-capturing substitution of meta-variables and capture-avoiding substitution of object variables. Further, solutions to unification constraints may also depend on freshness constraints, represented here by a predicate `not-free-in`.

Second-order matching was proved decidable by (Huet and Lang, 1978) and generalised in (Dowek, 1991, 1994; Padovani, 2000). In (Goldfarb, 1981), it was shown that second-order unification is also undecidable, by reduction to Hilbert's tenth problem. We have followed Goldfarb's methodology to provide the proof of undecidability for extended nominal unification. In (Levy, 1996), some cases of *linear* second-order unification are proved to be decidable. Particularly the case where each variable occurs at most twice. In (Levy, 1998; Levy and Veanes, 2000) it was proved that, under the same restriction, second-order unification is undecidable, along with other decidable and undecidable subclasses of second-order unification problems with variable occurrence restrictions. Such results are obtained by means of reducing *simultaneous rigid E-unification* (Gallier et al., 1987), proved to be undecidable by (Degtyarev and Voronkov, 1996), to second-order unification. Additionally in (Levy and Veanes, 2000), there is an undecidability proof of second-order unification by a direct encoding from the halting problem for Turing machines.

Extended Nominal terms Nominal terms were introduced by Urban et al. in (Urban et al., 2004). In extended nominal terms, the notion of higher-order substitution is inductively defined on nominal terms by means of a term-former and α-conversion is formalised using swappings and the freshness relation. This approach makes rewriting with extended nominal terms simpler and more readable than higher-order rewrite formalisms where substitution is implemented by de Bruijn indices, as is typical with explicit substitution systems, like in the case of (Bonelli et al., 2001) where Higher-Order Rewrite Systems are translated to first-order systems so that techniques and results from first-order can be transferred to the higher-order

formalism, or Pagano's Explicit Reduction Systems (Pagano, 1998) (XRSs) which are higher-order rewriting systems in a first-order framework combined with an explicit substitution calculus. Our theory of extended nominal terms follows closely that in (Fairweather et al., 2015) with some minor changes in the notation used for moderated variables. Also, we define only one freshness inference rule for variables, $(\#_{\mathbf{alt}}\mathbf{X})$, (see Definition 2.4.1) and introduce the freshness context as an argument of the disagreement set, $(\approx\alpha_{\mathbf{alt}}\mathbf{X})$, (see Definition 2.4.4) (in (Fairweather et al., 2015) the freshness context was not an argument of the disagreement set but this was a typo). However, these modifications are only presentational and both theories are equivalent. Also, their results are on dependent types for nominal terms while ours extend NRSs (Fernández and Gabbay, 2007; Fernández et al., 2004) and nominal unification. Merging both results is of great interest towards the development of a nominal logical framework. Capture-avoiding substitution using nominal techniques was already studied by Gabbay in (Gabbay, 2009; Gabbay and Mathijssen, 2008) for the language of Fraenkel-Mostowski set theory and for nominal algebra, which is an account of equational logic in the context of nominal sets. Permissive nominal terms (Dowek et al., 2010) are a variant of nominal terms which elide the explicit freshness relation by labelling variables with a set of fresh atoms and a set of free atoms as is expected to occur inside the variable. Hamana's Binding Term Rewriting Systems (Hamana, 2006) (BTRSs) is an extension of TRSs with variable binding based on Fiore et al. *abstract syntax with variable binding* (Fiore et al., 1999) which was presented at the same time as Gabbay and Pitts nominal abstract syntax in (Gabbay and Pitts, 2002) and have strong similarities with eNRSs at the level of syntax and the theories of the language. BTRSs add explicit renaming operations and restricts which terms may be substituted for a variable by denoting the set of free names occurring in such terms, known as *containment*, as opposed as the nominal approach where one provides the free names that must be avoided by such terms. Variables and names in BTRSs correspond to variables and atoms in eNRSs. Fiore and Staton define in (Fiore and Staton, 2014) a typed metalanguage based on the theory of capture-avoiding substitution demonstrating that substituting corresponds to jumping in an abstract machine (that is, calling, or returning from, a procedure). Then, computational effects for functional programming languages can be studied by means of the substitution theory. In explicit substitutions for Contextual Type Theory (Abel and Pientka, 2010), Abel and Pientka present an explicit substitution calculus for the logic given in (Nanevski et al., 2008) which distinguishes between variables, which can be bound, and meta-variables, which cannot be bound, and give an algorithm for definitional equality. Their notion of variable and meta-variable corresponds to atoms and variables in eNRSs respectively. Also, similarly to extended nominal terms, meta-variable substitution is possibly-capturing whereas variable substitution avoids capture. The

evaluation of both substitutions are treated lazily, their strategy to postpone the computation of substitutions until necessary. Then, equality of meta-variables is checked by making sure that their respective environments, that is, substitutions, are also equal. This is similar to checking α-equality of atom actions when deriving equality for nominal variables. Our theory does not have a typing system, however, in (Fairweather et al., 2015) a dependent type system was defined for extended terms and our treatment of variable binding avoids the use of de Bruijn indices when evaluating substitution applications. Nominal rewriting has also been extended in (Fernández and Gabbay, 2005) and (Gabbay, 2007), with machinery for name generation the former and with *hierarchical nominal terms* the latter.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

My thesis was that

> *Unification of nominal terms with atom substitution is undecidable. Nevertheless, the matching process is both decidable and unitary for a particular class of extended terms thus providing a means to mechanise the rewriting process for the extended nominal framework. The extension of nominal rewriting is then used to demonstrate there is a direct correspondence between Combinatory Reduction Systems and extended Nominal Rewrite Systems.*

To support my thesis I have defined two pairs of translation functions to provide a concrete correspondence between Combinatory Reduction Systems and the nominal rewriting framework, demonstrating that, despite their differences in the meta-language, it is possible to have a translation between these formalisms that preserves and reflects the rewrite relation, which is key to the translation of properties such as confluence and termination. Although previous work has been done on translating between nominal abstract and higher-order syntax specifications (Gacek, 2010) and nominal and higher-order pattern unification (Cheney, 2005; Levy and Villaret, 2012), our work differs by focusing on a syntax directed mapping of standard (extended) NRS rules and ground terms to CRS rules and terms and vice versa, providing a mechanism to export results from one rewriting framework to the other, for instance, due to the good algorithmic properties of nominal terms, CRS could be encoded into the nominal formalism to take advantage of existing nominal procedures such as orderings or completion (Fernández and Gabbay, 2010).

The first pair of translation functions provides a one-step reduction preserving translation from the class of closed nominal rules and ground terms to CRS rules and terms and a

reduction preserving translation from CRS to NRS where a set of nominal rules simulating higher-order substitution is added to each encoded CRS system to evaluate function application. This means that one rewrite step in the CRS system is simulated by one or more steps in the NRS system.

The second pair of functions encodes, CRS into an extension of NRS with implicit substitution and a class of closed-simple extended nominal rules and ground terms into CRS. The CRS to eNRS translation function was built on top of the CRS to NRS function by adapting the case for variables so that the function generates implicit substitutions instead of explicit ones. Then, proofs and properties from the original function were easily updated to the extended translation as a result of Theorem 2.3.6 stating the correspondence between the behaviour of explicit and implicit a-substitutions as given by the set of rules and the theory of a-substitution action respectively. The second translation function, that is, from the class of standard eNRS and ground terms to CRS was not directly encoded into CRSs, instead, I provided first a reduction-preserving translation of standard eNRS to NRS with the addition of function symbol `sub` (and the set of rules that describes the semantics of `sub`). To my knowledge, this is the first result relating implicit and explicit substitution in the nominal rewriting framework. Then, the NRS encoding was translated to CRS along with the additional set of rules for higher-order substitution. Accordingly, the translation function from the class of standard eNRS to CRS preserves but does not reflect the one-step rewrite relation for the general case.

The extended nominal rewriting framework is built on top of the nominal rewriting framework given in (Fernández and Gabbay, 2007). However, the unification algorithm given in (Urban et al., 2004) was no longer suitable as a matching tool for the extended rewriting theory I was defining. As a result, I provided a theory for extended nominal unification and characterised the unification problem for extended nominal terms. Then, it was shown that nominal unification for extended nominal terms is undecidable in general, by reducing Hilbert's tenth problem to extended nominal unification. Hilbert's tenth problem was proved undecidable by (Matiyasevich, 1970). Therefore, I defined a naive matching algorithm which produces the set of all correct solutions and only correct solutions to a (unifiable) matching problem. This was achieved by applying a syntactic restriction to such matching algorithm so that instantiation is trivial on one side of the equation. However, it was found that the property of unique most general solution could not be enjoyed for the general case since A matching constraint of form $[a \mapsto s]\cdot X \stackrel{?}{\approx} t$ has a high branching factor because there is a candidate solution for each position p in t since it must be checked whether s unifies with any subterm of t when generating substitutions. Although in this thesis my interest is not in algorithm complexity, it is observable that such high branching factor may render

the matching algorithm as impractical. Nevertheless, although the matching algorithm is not practical to automate rewriting, it is useful to provide an operational definition of the property of closedness in extended nominal terms due to the lack of a structural definition as the one given by Clouston in (Clouston, 2007) for non-extended terms. The matching algorithm was then further restricted following Fairweather's Phd thesis (Fairweather, 2014) and the TLCA 2015 paper on dependent types for extended nominal terms (Fairweather et al., 2015) design implementation by disallowing the generation of v-substitutions for matching constraints where the pattern term is a variable occurrence with non-trivial suspended a-substitutions. Then, the class of simple pattern matching problems was characterised, including only matching constraints which have at least one fixed variable occurrence with trivial a-substitutions in the pattern term, for each variable symbol in the constraint. It was then proven that for this particular class of matching constraints, the simple-matching algorithm finds at most one principal solution. The restrictions imposed to the class of simple pattern matching problems shows the intricacy that involves inducing a well-behaved rewrite relation when allowing a-substitution in patterns of rewrite rules. Another factor increasing the complexity of rewriting is handling equivariance when matching with rule patterns containing unabstracted atoms. In Chapter 6, a formal definition of extended nominal rewriting was given, first describing elementary rewriting for the extended framework where a meta-permutation was included in the generation of the rewrite relation to deal with equivariance in patterns of rules. Equivariance unification has been found to be NP-hard already for non-extended terms (Cheney, 2010), although in the case of equivariant matching without swappings on the pattern term, the complexity is polynomial (see also (Cheney, 2010)). However, since CRS rules are closed by definition, I did not need to provide an equivariant matching algorithm in order to prove the claim stated in my thesis. The definition of elementary rewriting was instead included as a stepping stone towards closed rewriting in the extended formalism where it was proved the rewrite relation satisfies the properties of its analogue in TRSs (i.e., closure under context application, closure under v-substitution) and NRSs (i.e., closure under permutation). Combining elementary rewriting, the simple-matching algorithm and the class of simple pattern matching problems I provided a definition of closed rewriting for extended terms by identifying the class of closed-simple terms. This definition was then used to prove the statement of my thesis along with the aforementioned one-step reduction-preserving translation functions.

9.2 Future Work

We now have a tool to transfer results between CRS and nominal rewriting. This could lead to procedures of nominal systems, for instance nominal orderings (Fernández and Rubio, 2012) being adapted to suit CRS or creation of new procedures by combination of existing ones from both formalism. Nominal typing systems (e.g. (Fairweather et al., 2011) for NRSs and (Fairweather et al., 2015) for eNRSs) could also be adapted to the (untyped) CRSs. It was also mentioned that one could benefit from a translation from standard eNRSs to CRSs that preserves the one-step rewrite relation, along with a structural description of closedness for extended nominal terms similar to the one given in (Clouston, 2007) for its non-extended analogue. Such structural definition is required to provide proofs of preservation of closedness when translating directly standard eNRSs to CRSs. Another interesting theoretical result left for future work is the definition of an encoding from eNRSs to NRSs (that is, allowing non-trivial a-substitutions on rule patterns). Such an encoding must delay the matching of moderated variables with non-trivial a-substitutions since the translation of implicit substitution to explicit by means of term-former `sub` modifies the structure of the term and thus cannot be applied on rule patterns because of the matching process. An approach to dealing with this issue is to replace each variable occurrence with non-trivial a-substitutions, $\phi\hat{}\pi\cdot X$, from a rule pattern l with a new variable symbol X' and check whether the matching is successful on the next rewrite step (if any) by using a term-former to relate each pair $(\phi\hat{}\pi\cdot X, X')$, for instance $\texttt{match}(\cdot,\cdot)$ and rule $\texttt{match}(X,X) \to \top$. Then, the right-hand side of the rule, r, is only activated if there is a result of $\top$ for every generated `match`.

Fairweather (Fairweather, 2014) and Fairweather, Fernández, Szasz and Tasistro (Fairweather et al., 2015) extended the syntax of nominal terms to include implicit substitution and dependent types with a view to define a *nominal logical framework*. A combination of the work done in this thesis extending NRSs and their work on dependent types is another step forwards towards building such logical framework.

One of the motivation points to extend nominal terms with implicit substitutions was its use as part of the logic programming language α-Prolog, as previously stated by Cheney in (Cheney, 2004b) and Cheney and Urban in (Cheney and Urban, 2004, 2008). The combination of a typing system for extended nominal terms, as pointed out above, and a unification procedure for such terms opens a path of research to extend α-Prolog with implicit substitution capabilities.

A semi-decision unification algorithm for extended nominal terms can be easily generalised from the matching algorithm given in Definition 5.2.6 by following the seminal work by Huet in (Huet, 1975). This involves discarding the syntactic restriction imposed to

unification constraints in our general matching algorithm as well as discarding rule $({}_?{\approx}\mathbf{XY})$, what is referred to as pre-unification procedure in higher-order unification. The main idea here is that constraints of form $\phi\hat{}\pi{\cdot}X \;{}_?{\approx}_?\; \phi'\hat{}\pi'{\cdot}Y$, known as flexible-flexible in (Huet, 1975), are always unifiable but have a high branching factor therefore they can be solved when a solution has already been found. An alternative approach for treating higher-order unification problems is (Dowek et al., 2000), based on explicit substitution calculi and develop over the $\lambda\sigma$-calculus. Calculi of explicit substitutions are essentially formal mechanisms attempting to solve the implicitness of substitution on the λ-calculus. Essentially, (Dowek et al., 2000) consists of, firstly, translating higher-order unification problems to the language of the explicit substitutions calculus; this process is known as a precooking translation. Afterwards, precooked problems are resolved as first-order unification problems modulo the equational theory which defines the calculus of explicit substitutions and then, the solutions are translated back to the language of the original problem. Therefore, the main advantage of the use of explicit substitutions is that the substitution operation becomes a first order substitution and higher-order substitutions can be obtained by applying the inverse of the precooking translation to the generated first order substitution. This approach has some similarities with our work on extended nominal terms and unification and in (Dowek et al., 2000) it was noted that their unification algorithm is a generalisation of Huet's method. Then, we leave the possibility of constructing a unification procedure and generalising such procedure based on the work in (Dowek et al., 2000) as future work.

It is of interest to find better suited grammar and representation of permutations and a-substitutions for extended nominal terms to look at the time and space complexity of the algorithms defined in this thesis following previous work on the matter by Calvès in (Calvès, 2010), Calvès and Fernández in (Calvès and Fernández, 2007, 2008a,b; Calvès and Fernández, 2009) and Levy and Villaret in (Levy and Villaret, 2010).

Bibliography

Abadi, M., Cardelli, L., Curien, P., and Lévy, J. (1991). Explicit substitutions. *J. Funct. Program.*, 1(4):375–416.

Abel, A. and Pientka, B. (2010). Explicit substitutions for contextual type theory. In *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010.*, pages 5–20.

Aczel, P. (1978). A general church-rosser theorem. Technical report, University of Manchester, Manchester.

Anantharaman, S., Lin, H., Lynch, C., Narendran, P., and Rusinowitch, M. (2010). Cap unification: Application to protocol security modulo homomorphic encryption. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 192–203, New York, NY, USA. ACM.

Arts, T. and Giesl, J. (2000). Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1):133 – 178.

Asperti, A. and Laneve, C. (1993). Interaction systems. In *Higher-Order Algebra, Logic, and Term Rewriting, First International Workshop, HOA '93, Amsterdam, The Netherlands, September 23-24, 1993, Selected Papers*, pages 1–19.

Ayala-Rincón, M., Fernández, M., Gabbay, M. J., and Oliveira, A. C. R. (2016). Checking overlaps of nominal rewriting rules. *Electr. Notes Theor. Comput. Sci.*, 323:39–56.

Baader, F. and Nipkow, T. (1998). *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA.

Baader, F. and Snyder, W. (2001). Unification theory. In Robinson, J. and Voronkov, A., editors, *Handbook of Automated Reasoning*, volume I, pages 447–533. Elsevier Science Publishers.

Barendregt, H. P. (1984). *The lambda calculus, its syntax and semantics*, volume 103. North-Holland, revised edition.

Bengtson, J. and Parrow, J. (2009). Formalising the pi-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2).

Bertolissi, C., Cirstea, H., and Kirchner, C. (2006). Expressing combinatory reduction systems derivations in the rewriting calculus. *Higher-Order and Symbolic Computation*, 19(4):345–376.

Bertolissi, C. and Kirchner, C. (2007). The rewriting calculus as a combinatory reduction system. In *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings*, pages 78–92.

Blanqui, F. (2006). Termination and confluence of higher-order rewrite systems. *CoRR*, abs/cs/0610064.

Bonelli, E., Kesner, D., and Ríos, A. (2001). From higher-order to first-order rewriting. In *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings*, pages 47–62.

Calvès, C. (2010). *Complexity and Implementation of Nominal Algorithms*. Phd thesis, King's College London. PhD thesis, King's College London.

Calvès, C. (2013). Unifying Nominal Unification. In van Raamsdonk, F., editor, *24th International Conference on Rewriting Techniques and Applications (RTA 2013)*, volume 21 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 143–157, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Calvès, C. and Fernández, M. (2007). Implementing nominal unification. *Electr. Notes Theor. Comput. Sci.*, 176(1):25–37.

Calvès, C. and Fernández, M. (2008a). Nominal matching and alpha-equivalence. In *Logic, Language, Information and Computation, 15th International Workshop, WoLLIC 2008, Edinburgh, UK, July 1-4, 2008, Proceedings*, pages 111–122.

Calvès, C. and Fernández, M. (2008b). A polynomial nominal unification algorithm. *Theor. Comput. Sci.*, 403(2-3):285–306.

Calvès, C. and Fernández, M. (2009). Matching and alpha-equivalence check for nominal terms. *Journal of Computer and System Sciences*. Special issue: Selected papers from WOLLIC 2008.

Calvès, C. and Fernández, M. (2010). The first-order nominal link. In *Logic-Based Program Synthesis and Transformation - 20th International Symposium, LOPSTR 2010, Hagenberg, Austria, July 23-25, 2010, Revised Selected Papers*, pages 234–248.

Cheney, J. (2004a). The complexity of equivariant unification. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 332–344.

Cheney, J. (2004b). *Nominal Logic Programming*. PhD thesis, Cornell University, Ithaca, NY.

Cheney, J. (2005). Relating nominal and higher-order pattern unification. In *Proceedings of UNIF 2005*, pages 104–119.

Cheney, J. (2010). Equivariant unification. *J. Autom. Reasoning*, 45(3):267–300.

Cheney, J. and Urban, C. (2004). αprolog: A logic programming language with names, binding and α-equivalence. In *Logic Programming*, pages 269–283. Springer Berlin Heidelberg.

Cheney, J. and Urban, C. (2008). Nominal logic programming. *ACM Trans. Program. Lang. Syst.*, 30(5):26:1–26:47.

Cirstea, H. and Kirchner, C. (1998). ρ-calculus. Its Syntax and Basic Properties. In *Workshop CCL'98*, Jerusalem, Israel.

Cirstea, H. and Kirchner, C. (2001a). The rewriting calculus — Part I. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–463.

Cirstea, H. and Kirchner, C. (2001b). The rewriting calculus - Part II. *IGPL*, 9(3):377–410.

Clouston, R. A. (2007). Closed terms. Available from http://users.cecs.anu.edu.au/~rclouston/closedterms.pdf.

Degtyarev, A. and Voronkov, A. (1996). The undecidability of simultaneous rigid e-unification. *Theor. Comput. Sci.*, 166(1&2):291–300.

Dershowitz, N. (1987). Termination of rewriting. *Journal of Symbolic Computation*, 3(1):69 – 115.

Dershowitz, N. (1989). Completion and its applications. In Aït-Kaci, H. and Nivat, M., editors, *Rewriting Techniques*, pages 31 – 85. Academic Press.

Dershowitz, N. and Jouannaud, J.-P. (1990). Rewrite systems. In *Handbook of Theoretical Computer Science*, volume B, pages 243–320.

Dershowitz, N. and Manna, Z. (1979). Proving termination with multiset orderings. *Commun. ACM*, 22(8):465–476.

Domínguez, J. (2014). A tool to translate between closed NRSs and CRSs. Available from http://www.inf.kcl.ac.uk/pg/domijesu/NRS2CRS.tar.gz.

Domínguez, J. and Fernández, M. (2014). Relating nominal and higher-order rewriting. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, pages 244–255.

Domínguez, J. and Fernández, M. (2015). From nominal to higher-order rewriting and back again. *Logical Methods in Computer Science*, 11(4).

Dowek, G. (1991). A second-order pattern matching algorithm for the cube of typed lambda-calculi. In *Mathematical Foundations of Computer Science 1991, 16th International Symposium, MFCS'91, Kazimierz Dolny, Poland, September 9-13, 1991, Proceedings*, pages 151–160.

Dowek, G. (1994). Third order matching is decidable. *Ann. Pure Appl. Logic*, 69(2-3):135–155.

Dowek, G., Gabbay, M. J., and Mulligan, D. P. (2010). Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Logic Journal of the IGPL*, 18(6):769–822.

Dowek, G., Hardin, T., and Kirchner, C. (1995). Higher-order unification via explicit substitutions (extended abstract). In *Proceedings, 10th Annual IEEE Symposium on Logic in Computer Science, San Diego, California, USA, June 26-29, 1995*, pages 366–374.

Dowek, G., Hardin, T., and Kirchner, C. (2000). Higher order unification via explicit substitutions. *Inf. Comput.*, 157(1-2):183–235.

Dowek, G., Hardin, T., Kirchner, C., and Pfenning, F. (1996). Unification via explicit substitutions: The case of higher-order patterns. In *Logic Programing, Proceedings of the 1996 Joint International Conference and Syposium on Logic Programming, Bonn, Germany, September 2-6, 1996*, pages 259–273.

Fairweather, E. (2014). *Type Systems for Nominal Terms*. PhD thesis, King's College London.

Fairweather, E., Fernández, M., and Gabbay, M. J. (2011). Principal types for nominal theories. *Lecture Notes in Computer Science*, 6914 LNCS:160–172.

Fairweather, E., Fernández, M., Szasz, N., and Tasistro, A. (2015). Dependent Types for Nominal Terms with Atom Substitutions. In Altenkirch, T., editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 180–195, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Fernández, M. and Gabbay, M. (2007). Nominal rewriting. *Inf. Comput.*, 205(6):917–965.

Fernández, M. and Gabbay, M. J. (2005). Nominal rewriting with name generation: Abstraction vs. locality. In *Proceedings of the 7th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '05, pages 47–58, New York, NY, USA. ACM.

Fernández, M. and Gabbay, M. J. (2010). Closed nominal rewriting and efficiently computable nominal algebra equality. In *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010.*, pages 37–51.

Fernández, M., Gabbay, M. J., and Mackie, I. (2004). Nominal rewriting systems. PPDP '04, pages 108–119, New York, NY, USA. ACM.

Fernández, M. and Rubio, A. (2012). Nominal completion for rewrite systems with binders. In Czumaj, A., Mehlhorn, K., Pitts, A., and Wattenhofer, R., editors, *Automata, Languages, and Programming*, volume 7392, pages 201–213. Springer Berlin Heidelberg.

Fiore, M. P., Plotkin, G. D., and Turi, D. (1999). Abstract syntax and variable binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202.

Fiore, M. P. and Staton, S. (2014). Substitution, jumps, and algebraic effects. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 41:1–41:10.

Gabbay, M. (2007). Hierarchical nominal terms and their theory of rewriting. *Electr. Notes Theor. Comput. Sci.*, 174(5):37–52.

Gabbay, M. and Pitts, A. M. (2002). A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363.

Gabbay, M. J. (2009). A study of substitution, using nominal techniques and fraenkel–mostowksi sets. *Theoretical Computer Science*, 410(12–13):1159 – 1189.

Gabbay, M. J. and Mathijssen, A. (2008). Capture-avoiding substitution as a nominal algebra. *Formal Aspects of Computation*, 20(4-5):451–479.

Gacek, A. (2008). The abella interactive theorem prover (system description). In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, pages 154–161.

Gacek, A. (2010). Relating nominal and higher-order abstract syntax specifications. In *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 26-28, 2010, Hagenberg, Austria*, pages 177–186.

Gallier, J. H., Raatz, S., and Snyder, W. (1987). Theorem proving using rigid e-unification equational matings. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*, pages 338–346.

Glauert, J., Kesner, D., and Khasidashvili, Z. (2005). Expression reduction systems and extensions: An overview. In Middeldorp, A., van Oostrom, V., van Raamsdonk, F., and Vrijer, R., editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838, pages 496–553. Springer Berlin Heidelberg.

Goldfarb, W. D. (1981). The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13(2):225 – 230.

Goubault-Larrecq, J. and Mackie, I. (2001). *Proof Theory and Automated Deduction*. Applied Logic Series. Springer Netherlands.

Hamana, M. (2006). An initial algebra approach to term rewriting systems with variable binders. *Higher-Order and Symbolic Computation*, 19(2-3):231–262.

Hamana, M. (2010). Semantic labelling for proving termination of combinatory reduction systems. In Escobar, S., editor, *Functional and Constraint Logic Programming*, volume 5979, pages 62–78. Springer Berlin Heidelberg.

Huet, G. P. (1973). The undecidability of unification in third order logic. *Information and Control*, 22(3):257 – 267.

Huet, G. P. (1975). A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57.

Huet, G. P. (1980). Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821.

Huet, G. P. and Lang, B. (1978). Proving and applying program transformations expressed with second-order patterns. *Acta Inf.*, 11:31–55.

Jouannaud, J. and Okada, M. (1997). Abstract data type systems. *Theor. Comput. Sci.*, 173(2):349–391.

Jouannaud, J.-P. (2005). Higher-order rewriting: Framework, confluence and termination. In Middeldorp, A., van Oostrom, V., van Raamsdonk, F., and Vrijer, R., editors, *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838, pages 224–250. Springer Berlin Heidelberg.

Jouannaud, J. P. and Okada, M. (1991). A computation model for executable higher-order algebraic specification languages. In *Logic in Computer Science, 1991. LICS '91., Proceedings of Sixth Annual IEEE Symposium on*, pages 350–361.

Khasidashvili, Z. (1990). Expression reduction systems. In *Proc. of I. Vekua Institute of Applied Mathematics*, volume 36, pages 200–220.

Klop, J. W. (1980). *Combinatory reduction systems*. PhD thesis, Univ. Utrecht.

Klop, J. W., van Oostrom, V., and van Raamsdonk, F. (1993). Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121(1&2):279–308.

Knuth, D. E. and Bendix, P. (1970). Simple word problem in universal algebra. In Leech, J., editor, *Computational problems in abstract algebra*, pages 263–297. Pergamon Press.

Kop, C. (2012). *Higher-order Termination*. PhD thesis, VU University Amsterdam.

Kumar, R. and Norrish, M. (2010). (nominal) unification by recursive descent with triangular substitutions. In *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, pages 51–66.

Levy, J. (1996). Linear second-order unification. In *Rewriting Techniques and Applications, 7th International Conference, RTA-96, New Brunswick, NJ, USA, July 27-30, 1996, Proceedings*, pages 332–346.

Levy, J. (1998). Decidable and undecidable second-order unification problems. In *Rewriting Techniques and Applications, 9th International Conference, RTA-98, Tsukuba, Japan, March 30 - April 1, 1998, Proceedings*, pages 47–60.

Levy, J. and Veanes, M. (2000). On the undecidability of second-order unification. *Inf. Comput.*, 159(1-2):125–150.

Levy, J. and Villaret, M. (2010). An efficient nominal unification algorithm. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010, July 11-13, 2010, Edinburgh, Scottland, UK*, pages 209–226.

Levy, J. and Villaret, M. (2012). Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10:1–10:31.

Lévy, J.-J. (2007). Generalized finite developments. In *Essays in Honour of Gilles Kahn*. Cambridge University Press, 2009.

Martelli, A. and Montanari, U. (1982). An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282.

Matiyasevich, Y. V. (1970). Enumerable sets are diophantine (in russian). *Soviet Mathematical Doklady*, 191(2):279–282.

Mayr, R. and Nipkow, T. (1998). Higher-order rewrite systems and their confluence. *Theor. Comput. Sci.*, 192(1):3–29.

Miller, D. (1991a). A logic programming language with lambda-abstraction, function variables, and simple unification. In Schroeder-Heister, P., editor, *Extensions of Logic Programming*, volume 475, pages 253–281. Springer Berlin Heidelberg.

Miller, D. (1991b). Unification of simply typed lambda-terms as logic programming. In *In Eighth International Logic Programming Conference*, pages 255–269. MIT Press.

Nadathur, G. and Miller, D. (1988). An overview of λ-prolog. *5th International Logic Programming Conference*, pages 810–827. cited By (since 1996)86.

Nanevski, A., Pfenning, F., and Pientka, B. (2008). Contextual modal type theory. *ACM Trans. Comput. Log.*, 9(3):23:1–23:49.

Newman, M. H. A. (1942). On theories with a combinatorial definition of 'equivalence'. In *Annals of Mathematics*, number 43 in 2, pages 223–243.

Nickolas, P. and Robinson, P. J. (1996). The qu-prolog unification algorithm: Formalisation and correctness. *Theor. Comput. Sci.*, 169(1):81–112.

Nipkow, T. (1991). Higher-order critical pairs. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 342–349.

Nipkow, T. and Prehofer, C. (1998). Higher-order rewriting and equational reasoning. In Bibel, W. and Schmitt, P., editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations*, volume 8, pages 399–430. Kluwer.

Padovani, V. (2000). Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 10(3):361–372.

Pagano, B. (1998). X.R.S : Explicit reduction systems - A first-order calculus for higher-order calculi. In *Automated Deduction - CADE-15, 15th International Conference on Automated Deduction, Lindau, Germany, July 5-10, 1998, Proceedings*, pages 72–87.

Paterson, M. and Wegman, M. (1978). Linear unification. *Journal of Computer and System Sciences*, 16(2):158 – 167.

Paulson, L. C. (1989). The foundation of a generic theorem prover. *J. Autom. Reasoning*, 5(3):363–397.

Paulson, L. C. (1994). *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*, volume 828 of *Lecture Notes in Computer Science*. Springer.

Pfenning, F. (1991). Logic programming in the LF logical framework. In Huet, G. and Plotkin, G., editors, *Logical Frameworks*, page 149–181. Cambridge University Press.

Pietrzykowski, T. (1973). A complete mechanization of second-order type theory. *J. ACM*, 20(2):333–364.

Pitts, A. M. (2003). Nominal logic, a first order theory of names and binding. *Inf. Comput.*, 186(2):165–193.

Pitts, A. M. (2013). *Nominal sets: names and symmetry in computer science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Rose, K. H. (1996). *Combinatory Reduction Systems with Extensions*. PhD thesis, University of Copenhagen.

Rose, K. H. (2011). CRSX - combinatory reduction systems with extensions. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, pages 81–90.

Silverman, J. H. (1997). *A Friendly Introduction to Number Theory*. Pearson, 4th edition 2012 edition.

Steinbach, J. (1994). *Termination of rewriting - extensions, comparison and automatic generation of simplification orderings*. PhD thesis, Universität Kaiserslautern.

Tait, W. W. (1967). Intensional interpretations of functionals of finite type I. *J. Symb. Log.*, 32(2):198–212.

Takahashi, M. (1993). Lambda-calculi with conditional rules. In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, pages 406–417.

Urban, C., Pitts, A. M., and Gabbay, M. (2004). Nominal unification. *Theor. Comput. Sci.*, 323(1-3):473–497.

van Oostrom, V. (1994). *Confluence for abstract and higher-order rewriting*. PhD thesis, VU University Amsterdam.

van Oostrom, V. and van Raamsdonk, F. (1994). Comparing combinatory reduction systems and higher-order rewrite systems. In Heering, J., Meinke, K., Möller, B., and Nipkow, T., editors, *Higher-Order Algebra, Logic, and Term Rewriting*, volume 816, pages 276–304. Springer Berlin Heidelberg.

van Raamsdonk, F. (1996). *Confluence and Normalization for Higher-Order Rewriting*. PhD thesis, Vrije Universiteit, Netherlands.

van Raamsdonk, F. (1999). Higher-order rewriting. In *Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings*, pages 220–239.

van Raamsdonk, F. (2015). Examples of higher-order rewriting. Available from http://www.cs.vu.nl/~femke/papers.html.

Wolfram, D. A. (1993). *The Clausal Theory of Types*. Cambridge University Press, New York, NY, USA.

Appendix A

Additional Proofs of Part II

Proof of Lemma 2.3.5.

Let t, s be CRS terms (and therefore also ground nominal terms (see (Domínguez and Fernández, 2014, Property 3.10))). Then, $t\phi \approx_\alpha t\{a \mapsto s\}$ where $t\{a \mapsto s\}$ denotes the term obtained by substituting (using the capture-avoiding substitution of the CRS) a by s in t.

Proof. By induction on the length of term t. Below, we denote structural congruence of CRS terms by $\equiv$.

- The case ($t = a$). Then, by application of Definition 2.3.2 we have, $a\phi \approx_\alpha s$ and by higher-order substitution we have, $a\{a \mapsto s\} \equiv s$. The property holds.

- The case ($t = b$). Then, by application of Definition 2.3.2 we have, $b\phi \approx_\alpha b$ and by higher-order substitution we have, $b\{a \mapsto s\} \equiv b$. The property holds.

- The case ($t = [b]t'$). Then, by application of Definition 2.3.2 we have, $\nabla \vdash ([b]t')\phi \approx_\alpha [c]((b\ c)\cdot t'\phi^{-c})$ where $\vdash c\#t', \mathscr{I}mg(())\phi)$ and, by higher-order substitution we have, $([b]t')\{a \mapsto s\} \equiv [b](t'\{a \mapsto s\})$ where $a \not\equiv b$ is ensured by α-conversion. The result follows by inductive hypothesis on $[c]((b\ c)\cdot t'\phi^{-c}) \approx_\alpha [b](t'\{a \mapsto s\})$.

- The case ($t = ft'$). Then, by application of Definition 2.3.2 we have, $(ft')\phi \approx_\alpha ft'\phi$ and by higher-order substitution we have, $(ft')\{a \mapsto s\} \equiv ft'\{a \mapsto s\}$. The result follows by inductive hypothesis on $f((b\ c)\cdot t'\phi \approx_\alpha ft'\{a \mapsto s\}$.

- The case ($t = (t_1, \ldots, t_n)$). Then, by application of Definition 2.3.2 we have, $(t_1, \ldots, t_n)\phi \approx_\alpha (t_1\phi, \ldots, t_n\phi)$ and by higher-order substitution we have, $(t_1, \ldots, t_n)\{a \mapsto s\} \equiv (t_1\{a \mapsto s\}, \ldots, t_n\{a \mapsto s\})$. The result follows by inductive hypothesis on $(t_1\phi, \ldots, t_n\phi) \approx_\alpha (t_1\{a \mapsto s\}, \ldots, t_n\{a \mapsto s\})$.

■

Proof of Theorem 2.5.8.

$\approx_\alpha$ *is an equivalence relation on nominal terms since*

- $\nabla \vdash s \approx_\alpha s$ *(reflexive),*
- *if* $\nabla \vdash s \approx_\alpha t$ *then* $\nabla \vdash t \approx_\alpha s$ *(symmetric) and*
- *if* $\nabla \vdash s \approx_\alpha t$ *and* $\nabla \vdash t \approx_\alpha u$ *, then* $\nabla \vdash s \approx_\alpha u$ *(transitive).*

Proof. The symmetric case.

By induction on the derivation of $\nabla \vdash s \approx_\alpha t$.

- The case ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{a}$). Using ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{a}$) we obtain $\nabla \vdash a \approx_\alpha a$ always.
- The case ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{X}$). Suppose $a\#X \in \nabla$ for all $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ so that $\nabla \vdash \phi\hat{}\pi \cdot X \approx_\alpha \phi'\hat{}\pi' \cdot X$ is derivable using ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{X}$). By inductive hypothesis on the unpacked definition of $\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ we obtain $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi'(\pi'(a)) \approx_\alpha \phi(\pi(a))\}$ such that $\mathtt{ds}(\nabla, \phi'\hat{}\pi', \phi\hat{}\pi) = \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. The result follows by application of rule ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{X}$).
- The case ($\approx_{\alpha_{\mathbf{alt}}}[\mathbf{a}]$). Suppose $\nabla \vdash s \approx_\alpha t$. By inductive hypothesis, $\nabla \vdash t \approx_\alpha s$. Applying ($\approx_{\alpha_{\mathbf{alt}}}[\mathbf{a}]$) we derive $\nabla \vdash [a]s \approx_\alpha [a]t$. The result follows.
- The case ($\approx_{\alpha_{\mathbf{alt}}}[\mathbf{b}]$). Suppose $\nabla \vdash (a\ b)\cdot s \approx_\alpha t$ where $\nabla \vdash b\#s$. Using case (1) of Lemma 2.5.4 we have, $\nabla \vdash s \approx_\alpha (a\ b)\cdot t$. Applying Lemma 2.5.7 we obtain $\nabla \vdash b\#(a\ b)\cdot t$. Applying case (3) of Lemma 2.5.4, $\nabla \vdash a\#t$. By inductive hypothesis, $\nabla \vdash (a\ b)\cdot t \approx_\alpha s$. Using ($\approx_{\alpha_{\mathbf{alt}}}[\mathbf{b}]$), $\nabla \vdash [a]s \approx_\alpha [b]t$. The result follows.
- The case ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{f}$). Suppose $\nabla \vdash s \approx_\alpha t$. By inductive hypothesis, $\nabla \vdash t \approx_\alpha s$. Using ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{f}$), $\nabla \vdash fs \approx_\alpha ft$. The result follows.
- The case ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{tupl}$). Suppose $\nabla \vdash s_1 \approx_\alpha t_1, \ldots, \nabla \vdash s_n \approx_\alpha t_n$. By inductive hypothesis, $\nabla \vdash t_1 \approx_\alpha s_1, \ldots, \nabla \vdash t_n \approx_\alpha s_n$. Using ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{tupl}$), $\nabla \vdash (s_1, \ldots, s_n) \approx_\alpha (t_1, \ldots, t_n)$. The result follows.

The transitive case.

By induction on the derivation of $\nabla \vdash s \approx_\alpha t,\ t \approx_\alpha u$.

- The case ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{a}$). Suppose $\nabla \vdash a \approx_\alpha t$. By the structure of the derivation rules, $t = a$. Similarly, suppose $\nabla \vdash s \approx_\alpha a$. By the structure of the derivation rules, $s = a$. The result follows by rule ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{a}$).
- The case ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{X}$). Suppose $\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_t\hat{}\pi_t)\#X \subseteq \nabla, \mathtt{ds}(\nabla, \phi_t\hat{}\pi_t, \phi_u\hat{}\pi_u)\#X \in \nabla$. Then, both $\nabla \vdash \phi_s\hat{}\pi_s \cdot X \approx_\alpha \phi_t\hat{}\pi_t \cdot X$ and $\nabla \vdash \phi_t\hat{}\pi_t \cdot X \approx_\alpha \phi_u\hat{}\pi_u \cdot X$ are derivable by rule ($\approx_{\alpha_{\mathbf{alt}}}\mathbf{X}$). We must prove that $\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_u\hat{}\pi_u) \subseteq$

$(\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_t\hat{}\pi_t) \cup \mathtt{ds}(\nabla, \phi_t\hat{}\pi_t, \phi_u\hat{}\pi_u))$. This can be done by application of the inductive step on the unpacked definition of both $\mathtt{ds}(\nabla, \phi_s\hat{}\pi_s, \phi_t\hat{}\pi_t)$ and $\mathtt{ds}(\nabla, \phi_t\hat{}\pi_t, \phi_u\hat{}\pi_u)$, that is, $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi_s(\pi_s(a)) \approx_\alpha \phi_t(\pi_t(a))\}$ and $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi_t(\pi_t(a)) \approx_\alpha \phi_u(\pi_u(a))\}$, such that $\{a \mid a \in \mathscr{A} \wedge \nabla \not\vdash \phi_s(\pi_s(a)) \approx_\alpha \phi_u(\pi_u(a))\}$ is obtained. Then, the result follow by rule $(\approx_{\alpha\mathbf{alt}}\mathbf{X})$.

- The case $(\approx_{\alpha\mathbf{alt}}[\mathbf{a}])$. Suppose $\nabla \vdash s \approx_\alpha t$, $t \approx_\alpha u$. By inductive hypothesis, $\nabla \vdash s \approx_\alpha u$. The result follows by rule $(\approx_{\alpha\mathbf{alt}}[\mathbf{a}])$.

- The case $(\approx_{\alpha\mathbf{alt}}[\mathbf{b}])$. Suppose $\nabla \vdash (a\ b)\cdot s \approx_\alpha t$, $(b\ c)\cdot t \approx_\alpha u$, where $\nabla \vdash b\#s, c\#t$. Applying Lemma 2.5.7, $\nabla \vdash c\#(a\ b)\cdot s$ and by case (3) of Lemma 2.5.4, $\nabla \vdash c\#s$. By Lemma 2.5.5, $\nabla \vdash (b\ c)\cdot((a\ b)\cdot s) \approx_\alpha (b\ c)\cdot t$. Then, by inductive hypothesis we obtain $\nabla \vdash (b\ c)\cdot((a\ b)\cdot s) \approx_\alpha u$. By Definition 2.1.8, $(b\ c)\cdot((a\ b)\cdot s) = (b\ c)(a\ b)\cdot s$. Since $\mathtt{ds}(\nabla, (b\ c)(a\ b), (a\ c)) = \{b, c\}$ and $\nabla \vdash b, c\#s$, by application of Lemma 2.5.6 we obtain, $\nabla \vdash (a\ c)\cdot s \approx_\alpha (b\ c)(a\ b)\cdot s \approx_\alpha u$. Applying again the inductive step we derive $\nabla \vdash (a\ c)\cdot s \approx_\alpha u$. Hence, using $(\approx_{\alpha\mathbf{alt}}[\mathbf{b}])$, $\nabla \vdash [a]s \approx_\alpha [b]t$, $\nabla \vdash [b]t \approx_\alpha [c]u$. The result follows.

- The case $(\approx_{\alpha\mathbf{alt}}\mathbf{f})$. Suppose $\nabla \vdash s \approx_\alpha t$, $t \approx_\alpha u$. By inductive hypothesis, $\nabla \vdash s \approx_\alpha u$. The result follows by rule $(\approx_{\alpha\mathbf{alt}}\mathbf{f})$.

- The case $(\approx_{\alpha\mathbf{alt}}\mathbf{tupl})$. Suppose $\nabla \vdash s_i \approx_\alpha t_i$, $t_i \approx_\alpha u_i$ for $1 \leq i \leq n$. By inductive hypothesis, $\nabla \vdash s_i \approx_\alpha u_i$. The result follows by rule $(\approx_{\alpha\mathbf{alt}}\mathbf{tupl})$.

■

Proof of Lemma 2.5.9.

- $\nabla \vdash \pi\cdot(s\phi) \approx_\alpha (\pi\cdot s)(\pi\cdot\phi)$.

- $\nabla \vdash (\pi\cdot s)\phi \approx_\alpha \pi\cdot(s(\pi^{-1}\cdot\phi))$.

Proof.

- The case $(s = n)$. There are two cases to consider depending on whether $n = a$ or not.

  For the first claim.

  Suppose $n = a$. Then, regardless of whether $a \in \mathscr{S}upport(\pi)$ or not we obtain $\nabla \vdash \pi\cdot t \approx_\alpha \pi\cdot t$.

  Suppose $n \neq a$ and $n \notin \mathscr{S}upport(\pi)$. Then, $\nabla \vdash n \approx_\alpha n$. Suppose now $n \in \mathscr{S}upport(\pi)$. Then $\nabla \vdash \pi(n) \approx_\alpha \pi(n)$.

  Hence the property holds for this case.

For the second claim.

Suppose $n = a$. Then, if $a \in \mathscr{S}upport(\pi)$, $\nabla \vdash \pi(a) \approx_\alpha \pi(a)$. Otherwise, if $a \notin \mathscr{S}upport(\pi)$, we obtain $\nabla \vdash t \approx_\alpha (\pi \circ \pi^{-1}) \cdot t$. Applying claim (2) of Lemma 2.5.4, we get $\nabla \vdash t \approx_\alpha t$ and we are done.

Now, suppose $n \neq a$. Then, if $n \in \mathscr{S}upport(\pi)$ and $\pi(n) = a$ we have $\nabla \vdash t \approx_\alpha (\pi \circ \pi^{-1}) \cdot t$. Applying claim (2) of Lemma 2.5.4, $\nabla \vdash t \approx_\alpha t$. Otherwise, if $n \in \mathscr{S}upport(\pi)$ and $\pi(n) \neq a$, we have $\nabla \vdash \pi(n) \approx_\alpha \pi(n)$. For the case where $n \notin \mathscr{S}upport(\pi)$, we obtain $\nabla \vdash n \approx_\alpha n$.

Hence the property holds for this case.

- The case $(s = \phi \hat{} \pi' \cdot X)$.

  For the first claim.

  Suppose $\nabla \vdash \pi \cdot ((\phi' \hat{} \pi' \cdot X)\phi)$. By Definition 2.3.2, $\nabla \vdash \pi \cdot ((\phi' \hat{} \pi' \cdot X)\phi) \approx_\alpha \pi \cdot ((\phi' \bullet \phi) \hat{} \pi' \cdot X)$. By Definition 2.1.8, $\pi \cdot ((\phi' \bullet \phi) \hat{} \pi' \cdot X) = ((\pi \cdot \phi') \bullet (\pi \cdot \phi)) \hat{} (\pi \circ \pi') \cdot X$. Applying again Definition 2.3.2 we obtain $\nabla \vdash ((\pi \cdot \phi') \bullet (\pi \cdot \phi)) \hat{} (\pi \circ \pi') \cdot X \approx_\alpha ((\pi \cdot \phi') \hat{} (\pi \circ \pi') \cdot X)(\pi \cdot \phi)$. Then, by Definition 2.1.8 we have, $((\pi \cdot \phi') \hat{} (\pi \circ \pi') \cdot X)(\pi \cdot \phi) = (\pi \cdot (\phi' \hat{} \pi' \cdot X))(\pi \cdot \phi)$. The result follows by the transitive property from Theorem 2.5.8.

  For the second claim.

  Suppose $\nabla \vdash (\pi \cdot (\phi' \hat{} \pi' \cdot X))\phi$. By Definition 2.1.8 we have, $(\pi \cdot (\phi' \hat{} \pi' \cdot X))\phi = ((\pi \cdot \phi') \hat{} (\pi \circ \pi') \cdot X)\phi$. By Definition 2.3.2 we have, $\nabla \vdash ((\pi \cdot \phi') \hat{} (\pi \circ \pi') \cdot X)\phi \approx_\alpha ((\pi \cdot \phi') \bullet \phi) \hat{} (\pi \circ \pi') \cdot X$. By Definition 2.1.8, $((\pi \cdot \phi') \bullet \phi) \hat{} (\pi \circ \pi') \cdot X = \pi \cdot ((\phi' \bullet (\pi^{-1} \cdot \phi)) \hat{} \pi' \cdot X)$. Applying again Definition 2.3.2 we obtain, $\nabla \vdash \pi \cdot ((\phi' \bullet (\pi^{-1} \cdot \phi)) \hat{} \pi' \cdot X) \approx_\alpha \pi \cdot ((\phi' \hat{} \pi' \cdot X)(\pi^{-1} \cdot \phi))$. The result follows by the transitive property from Theorem 2.5.8.

- The case $(s = [a]s')$.

  For the first claim.

  Suppose $\nabla \vdash \pi \cdot (((a\ c) \cdot s')\phi^{-c}) \approx_\alpha (\pi \cdot ((a\ c) \cdot s'))(\pi \cdot \phi^{-c})$ where $\nabla \vdash c \# s', \mathscr{I}mg(\phi)$. Using $(\approx_{\alpha_{\mathbf{alt}}}[\mathbf{a}])$ we obtain $\nabla \vdash [\pi(c)]\pi \cdot (((a\ c) \cdot s')\phi^{-c}) \approx_\alpha [\pi(c)](\pi \cdot ((a\ c) \cdot s'))(\pi \cdot \phi^{-c})$. By Definition 2.1.8 on the LHS we get, $[\pi(c)]\pi \cdot (((a\ c) \cdot s')\phi^{-c}) = \pi \cdot ([c]((a\ c) \cdot s')\phi^{-c})$ and on the RHS we have, $[\pi(c)](\pi \cdot ((a\ c) \cdot s'))(\pi \cdot \phi^{-c}) = [\pi(c)](\pi(a\ c) \cdot s')(\pi \cdot \phi^{-c})$. Now, since $\mathtt{ds}(\nabla, \pi(a\ c), (\pi(a)\ \pi(c))\pi) = \varnothing$ then, on the RHS we have $\nabla \vdash [\pi(c)](\pi(a\ c) \cdot s')(\pi \cdot \phi^{-c}) \approx_\alpha [\pi(c)]((\pi(a)\ \pi(c))\pi \cdot s')(\pi \cdot \phi^{-c})$. By Definition 2.3.2 on the LHS, $\nabla \vdash \pi \cdot ([c]((a\ c) \cdot s')\phi^{-c}) \approx_\alpha \pi \cdot (([a]s')\phi)$. By the transitive property

from Theorem 2.5.8, $\nabla \vdash \pi\cdot(([a]s')\phi) \approx_\alpha [\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')(\pi\cdot\phi^{-c})$. By Definition 2.3.2 on the RHS, $\nabla \vdash [\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')(\pi\cdot\phi^{-c}) \approx_\alpha ([\pi(a)]\pi\cdot s')(\pi\cdot\phi)$. By Definition 2.1.8 on the RHS, $([\pi(a)]\pi\cdot s')(\pi\cdot\phi) = (\pi\cdot([a]s'))(\pi\cdot\phi)$. The result follows by the transitive property from Theorem 2.5.8.

For the second claim.

Suppose $\nabla \vdash (\pi\cdot((a\ c)\cdot s'))\phi^{-c} \approx_\alpha \pi\cdot(((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c})$ where $\nabla \vdash c\#s', \mathscr{I}mg(\phi)$. Using $(\approx_{\alpha_{\mathbf{alt}}}[\mathbf{a}])$ we obtain, $\nabla \vdash [\pi(c)](\pi\cdot((a\ c)\cdot s'))\phi^{-c} \approx_\alpha [\pi(c)]\pi\cdot(((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c})$. On the LHS we have, by Definition 2.1.8, $[\pi(c)](\pi\cdot((a\ c)\cdot s'))\phi^{-c} = [\pi(c)](\pi(a\ c)\cdot s')\phi^{-c}$. By the property of permutations we obtain, $[\pi(c)](\pi(a\ c)\cdot s')\phi^{-c} =$ $[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')\phi^{-c}$ and by Definition 2.1.8, $[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')\phi^{-c} = [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot s'))\phi^{-c}$. By Definition 2.3.2 on the LHS we get $\nabla \vdash [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot s'))\phi^{-c} \approx_\alpha ([\pi(a)](\pi\cdot s'))\phi$ and by Definition 2.1.8, $([\pi(a)](\pi\cdot s'))\phi = \pi\cdot([a]s')\phi$. Now, on the RHS we have, by Definition 2.1.8, $[\pi(c)]\pi\cdot(((a\ c)\cdot s')\pi^{-1}\cdot\phi^{-c} = \pi\cdot([c]((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c}))$. By Definition 2.3.2, $\nabla \vdash \pi\cdot([c]((a\ c)\cdot s')(\pi^{-1}\cdot\phi^{-c}))$. The result follows by the transitive property from Theorem 2.5.8.

- The case $(s = [b]s')$.

  For the first claim.

  Assume there is $c \in \mathscr{A}$ such that $\nabla \vdash c\#s', \mathscr{I}mg(\phi)$. Suppose also that $\nabla \vdash \pi\cdot(((a\ c)\cdot s')\phi^{-c}) \approx_\alpha (\pi\cdot((a\ c)\cdot s'))(\pi\cdot\phi^{-c})$. Using $(\approx_{\alpha_{\mathbf{alt}}}[\mathbf{a}])$ we obtain $\nabla \vdash [\pi(c)]\pi\cdot(((a\ c)\cdot s')\phi^{-c}) \approx_\alpha [\pi(c)](\pi\cdot((a\ c)\cdot s'))(\pi\cdot\phi^{-c})$. On the LHS we have, by Definition 2.1.8, $[\pi(c)]\pi\cdot(((a\ c)\cdot s')\phi^{-c}) = \pi\cdot[c]((a\ c)\cdot s')\phi^{-c}$. On the RHS we have, by Definition 2.1.8, $[\pi(c)](\pi\cdot((a\ c)\cdot s'))(\pi\cdot\phi^{-c}) = [\pi(c)](\pi(a\ c)\cdot s')(\pi\cdot\phi^{-c})$. By the property of permutations we obtain, $[\pi(c)](\pi(a\ c)\cdot s')(\pi\cdot\phi^{-c}) =$ $[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')(\pi\cdot\phi^{-c})$ and by Definition 2.1.8, $[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot s')(\pi\cdot\phi^{-c}) = [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot s'))(\pi\cdot\phi^{-c})$. By Definition 2.3.2 on the LHS we obtain, $\nabla \vdash \pi\cdot[c]((a\ c)\cdot s')\phi^{-c} \approx_\alpha \pi\cdot(([a]s')\phi)$ and on the RHS we get $\nabla \vdash [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot s'))(\pi\cdot\phi^{-c}) \approx_\alpha ([\pi(a)]\pi\cdot s')(\pi\cdot\phi)$. By application of Definition 2.1.8 we have, $([\pi(a)]\pi\cdot s')(\pi\cdot\phi) = (\pi\cdot[a]s')(\pi\cdot\phi)$. The result follows by the transitive property from Theorem 2.5.8.

  For the second claim.

  The case is solved similarly to the previous case, $(s = [a]s')$, and thus omitted here.

- The case $(s = fs')$.

For the first claim.

Suppose $\nabla \vdash \pi{\cdot}(s'\phi) \approx_\alpha (\pi{\cdot}s')(\pi{\cdot}\phi)$. Using $(\approx\alpha_{\mathbf{alt}}\mathbf{f})$ we obtain $\nabla \vdash f\pi{\cdot}(s'\phi) \approx_\alpha f(\pi{\cdot}s')(\pi{\cdot}\phi)$.

On the LHS, By Definition 2.1.8, $f\pi{\cdot}(s'\phi) = \pi{\cdot}(fs'\phi)$. By Definition 2.3.2 we have $\nabla \vdash \pi{\cdot}(fs'\phi) \approx_\alpha \pi{\cdot}((fs')\phi)$.

On the RHS, By Definition 2.3.2 we have $\nabla \vdash f(\pi{\cdot}s')(\pi{\cdot}\phi) \approx_\alpha (f(\pi{\cdot}s'))(\pi{\cdot}\phi)$. By Definition 2.1.8, $(f(\pi{\cdot}s'))(\pi{\cdot}\phi) = (\pi{\cdot}fs')(\pi{\cdot}\phi)$. The result follows by the transitive property from Theorem 2.5.8.

For the second claim.

Suppose $\nabla \vdash (\pi{\cdot}s')\phi \approx_\alpha \pi{\cdot}(s'(\pi^{-1}{\cdot}\phi))$. Using $(\approx\alpha_{\mathbf{alt}}\mathbf{f})$ we obtain $\nabla \vdash f(\pi{\cdot}s')\phi \approx_\alpha f\pi{\cdot}(s'(\pi^{-1}{\cdot}\phi))$.

On the LHS, By Definition 2.3.2 we have $\nabla \vdash f(\pi{\cdot}s')\phi \approx_\alpha (f(\pi{\cdot}s'))\phi$ By Definition 2.1.8, $(f(\pi{\cdot}s'))\phi = (\pi{\cdot}fs')\phi$.

On the RHS, By Definition 2.1.8, $f\pi{\cdot}(s'(\pi^{-1}{\cdot}\phi)) = \pi{\cdot}(fs'(\pi^{-1}{\cdot}\phi))$. By Definition 2.3.2 we have $\nabla \vdash \pi{\cdot}(fs'(\pi^{-1}{\cdot}\phi)) \approx_\alpha \pi{\cdot}((fs')(\pi^{-1}{\cdot}\phi))$. The result follows by the transitive property from Theorem 2.5.8.

- The case $(s = (s_1, \ldots, s_n))$.

  For the first claim.

  Suppose $\nabla \vdash \pi{\cdot}(s_1\phi) \approx_\alpha (\pi{\cdot}s_1)(\pi{\cdot}\phi) \cdots \nabla \vdash \pi{\cdot}(s_n\phi) \approx_\alpha (\pi{\cdot}s_n)(\pi{\cdot}\phi)$. Using $(\approx\alpha_{\mathbf{alt}}\mathbf{tupl})$ we obtain, $\nabla \vdash (\pi{\cdot}(s_1\phi), \ldots, \pi{\cdot}(s_n\phi)) \approx_\alpha ((\pi{\cdot}s_1)(\pi{\cdot}\phi), \ldots, (\pi{\cdot}s_n)(\pi{\cdot}\phi))$.

  On the LHS, By Definition 2.1.8 we have, $(\pi{\cdot}(s_1\phi), \ldots, \pi{\cdot}(s_n\phi)) = \pi{\cdot}(s_1\phi, \ldots, s_n\phi)$. By Definition 2.3.2, $\nabla \vdash \pi{\cdot}(s_1\phi, \ldots, s_n\phi) \approx_\alpha \pi{\cdot}((s_1, \ldots, s_n)\phi)$.

  On the RHS, By Definition 2.3.2 we have, $\nabla \vdash ((\pi{\cdot}s_1)(\pi{\cdot}\phi), \ldots, (\pi{\cdot}s_n)(\pi{\cdot}\phi)) \approx_\alpha (\pi{\cdot}s_1, \ldots, \pi{\cdot}s_n)(\pi{\cdot}\phi)$. By Definition 2.1.8, $(\pi{\cdot}s_1, \ldots, \pi{\cdot}s_n)(\pi{\cdot}\phi) = (\pi{\cdot}(s_1, \ldots, s_n))(\pi{\cdot}\phi)$. The result follows by the transitive property from Theorem 2.5.8.

  For the second claim.

  Suppose $\nabla \vdash (\pi{\cdot}s_1)\phi \approx_\alpha \pi{\cdot}(s_1(\pi^{-1}{\cdot}\phi)) \cdots \nabla \vdash (\pi{\cdot}s_n)\phi \approx_\alpha \pi{\cdot}(s_n(\pi^{-1}{\cdot}\phi))$. Using $(\approx\alpha_{\mathbf{alt}}\mathbf{tupl})$ we obtain, $\nabla \vdash ((\pi{\cdot}s_1)\phi, \ldots, (\pi{\cdot}s_n)\phi) \approx_\alpha (\pi{\cdot}(s_1(\pi^{-1}{\cdot}\phi)), \ldots, \pi{\cdot}(s_n(\pi^{-1}{\cdot}\phi)))$.

  On the LHS, By Definition 2.3.2 we have, $\nabla \vdash ((\pi{\cdot}s_1)\phi, \ldots, (\pi{\cdot}s_n)\phi) \approx_\alpha (\pi{\cdot}s_1, \ldots, \pi{\cdot}s_n)\phi$. By Definition 2.1.8, $(\pi{\cdot}s_1, \ldots, \pi{\cdot}s_n)\phi = (\pi{\cdot}(s_1, \ldots, s_n))\phi$.

  On the RHS, By Definition 2.1.8, $(\pi{\cdot}(s_1(\pi^{-1}{\cdot}\phi)), \ldots, \pi{\cdot}(s_n(\pi^{-1}{\cdot}\phi))) \approx_\alpha \pi{\cdot}(s_1(\pi^{-1}{\cdot}\phi), \ldots, s_n(\pi^{-1}{\cdot}\phi))$. By Definition 2.3.2 we have

$\pi\cdot(s_1(\pi^{-1}\cdot\phi),\ldots,s_n(\pi^{-1}\cdot\phi)) \approx_\alpha \pi\cdot((s_1,\ldots,s_n)(\pi^{-1}\cdot\phi))$. The result follows by the transitive property from Theorem 2.5.8.

■

Proof of Property 2.5.11.

- $\nabla \vdash (s\sigma)\phi\sigma \approx_\alpha (s\phi)\sigma$.

Proof. By induction on the structure of s.

- The case $(s = a)$. By Definition 2.3.9, $a\sigma = a$. Then, we have $\nabla \vdash a(\phi\sigma) \approx_\alpha a(\phi\sigma)$ and thus it holds.

- The case $(s = \phi'^\frown\pi\cdot X)$. Suppose $\nabla \vdash ((\phi'^\frown\pi\cdot X)\sigma)\phi\sigma$. By Definition 2.3.9 we have, $\nabla \vdash ((\phi'^\frown\pi\cdot X)\sigma)\phi\sigma \approx_\alpha ((\pi\cdot\sigma(X))(\phi'\sigma))(\phi\sigma)$. By application of Definition 2.1.5, $((\pi\cdot\sigma(X))(\phi'\sigma))(\phi\sigma) = (\pi\cdot\sigma(X)(\phi'\sigma \bullet \phi\sigma)$. By application of Definition 2.3.2, $\nabla \vdash (\pi\cdot\sigma(X)(\phi'\sigma \bullet \phi\sigma) \approx_\alpha ((\phi'\sigma \bullet \phi\sigma)^\frown\pi\cdot\sigma(X)$. Using the transitive property from Theorem 2.5.8, $\nabla \vdash ((\phi'^\frown\pi\cdot X)\sigma)\phi\sigma \approx_\alpha ((\phi'\sigma \bullet \phi\sigma)^\frown\pi\cdot\sigma(X)$. By Definition 2.3.9, $\nabla \vdash ((\phi'\sigma \bullet \phi\sigma)^\frown\pi\cdot\sigma(X) \approx_\alpha ((\phi' \bullet \phi)^\frown\pi\cdot X)\sigma$. The result follows by Definition 2.3.2 and the transitive property from Theorem 2.5.8.

- The case $(s = [a]t)$. Suppose $\nabla \vdash (((a\ c)\cdot t)\sigma)(\phi^{-c}\sigma) \approx_\alpha (((a\ c)\cdot t)\phi^{-c})\sigma$. Using $(\approx_{\alpha_{\mathbf{alt}}[\mathbf{a}]})$ we derive $\nabla \vdash [c](((a\ c)\cdot t)\sigma)(\phi^{-c}\sigma) \approx_\alpha [c](((a\ c)\cdot t)\phi^{-c})\sigma$.

  On the LHS, by Definition 2.3.9 we obtain $[c](((a\ c)\cdot t)\sigma)(\phi^{-c}\sigma) = [c]((a\ c)\cdot t\sigma)(\phi^{-c}\sigma)$. Applying the definition again we have, $[c]((a\ c)\cdot t\sigma)(\phi^{-c}\sigma) = ([c]((a\ c)\cdot t\sigma)\phi^{-c})\sigma$. By Definition 2.3.2, $\nabla \vdash ([c]((a\ c)\cdot t\sigma)(\phi^{-c})\sigma) \approx_\alpha ([a]t\sigma)(\phi\sigma)$. By Definition 2.3.9, $([a]t\sigma)(\phi\sigma) = (([a]t)\sigma)(\phi\sigma)$.

  On the RHS, by Definition 2.3.9 we have $[c](((a\ c)\cdot t)\phi^{-c})\sigma = ([c]((a\ c)\cdot t)\phi^{-c})\sigma$. By Definition 2.3.2, $\nabla \vdash ([c]((a\ c)\cdot t)\phi^{-c})\sigma \approx_\alpha (([a]t)\phi^{-c})\sigma$.

  The result follows by the transitive property from Theorem 2.5.8.

- The case $(s = ft)$. Suppose $\nabla \vdash (t\sigma)(\phi\sigma) \approx_\alpha (t\phi)\sigma$. Using $(\approx_{\alpha_{\mathbf{alt}}\mathbf{f}})$ we derive $\nabla \vdash f(t\sigma)(\phi\sigma) \approx_\alpha f(t\phi)\sigma$.

  On the LHS, by Definition 2.3.9, $\nabla \vdash f(t\sigma)(\phi\sigma) = \nabla \vdash (f(t\sigma)\phi)\sigma$. By Definition 2.3.2, we obtain $\nabla \vdash (f(t\sigma)\phi)\sigma \approx_\alpha (ft\sigma)(\phi\sigma)$. By Definition 2.3.9, $(ft\sigma)(\phi\sigma) = ((ft)\sigma)(\phi\sigma)$.

  On the RHS, by Definition 2.3.9 we have $f(t\phi)\sigma = (ft\phi)\sigma$ By Definition 2.3.2, $\nabla \vdash (ft\phi)\sigma \approx_\alpha ((ft)\phi)\sigma$.

The result follows by the transitive property from Theorem 2.5.8.

- The case ($s = (t_1, \ldots, t_n)$). Suppose $\nabla \vdash (t_1\sigma)(\phi\sigma) \approx_\alpha (t_1\phi)\sigma \cdots \nabla \vdash (t_n\sigma)(\phi\sigma) \approx_\alpha (t_n\phi)\sigma$. Using ($\approx_{\alpha\,\mathbf{alt}}\mathbf{tupl}$) we derive $\nabla \vdash ((t_1\sigma)(\phi\sigma), \ldots, (t_n\sigma)(\phi\sigma)) \approx_\alpha ((t_1\phi)\sigma, \ldots, (t_n\phi)\sigma)$.

  On the LHS, by Definition 2.3.9, $((t_1\sigma)(\phi\sigma), \ldots, (t_n\sigma)(\phi\sigma)) = ((t_1\sigma)\phi, \ldots, (t_n\sigma)\phi)\sigma$. By Definition 2.3.2, we obtain $\nabla \vdash ((t_1\sigma)\phi, \ldots, (t_n\sigma)\phi)\sigma \approx_\alpha (t_1\sigma, \ldots, t_n\sigma)(\phi\sigma)$. By Definition 2.3.9, $(t_1\sigma, \ldots, t_n\sigma)(\phi\sigma) = ((t_1, \ldots, t_n)\sigma)(\phi\sigma)$.

  On the RHS, by Definition 2.3.9 we have $((t_1\phi)\sigma, \ldots, (t_n\phi)\sigma) = (t_1\phi, \ldots, t_n\phi)\sigma$. By Definition 2.3.2, $\nabla \vdash (t_1\phi, \ldots, t_n\phi)\sigma \approx_\alpha ((t_1, \ldots, t_n)\phi)\sigma$.

  The result follows by the transitive property from Theorem 2.5.8.

■

Proof of Lemma 2.5.12.

For any pair of freshness contexts Δ, ∇ and v-substitution σ such that $\Delta \vdash \nabla\sigma$,

1. *If $\nabla \vdash a\#t$ then $\Delta \vdash a\#t\sigma$, and*
2. *If $\nabla \vdash s \approx_\alpha t$ then $\Delta \vdash s\sigma \approx_\alpha t\sigma$.*

Proof. We handle both cases separately.

For the first claim.

By induction on the derivation of $\nabla \vdash a\#t$.

- The case ($\#_{\mathbf{alt}}\mathbf{a}$). By Definition 2.3.9, $b\sigma = b$, and the result follows.
- The case ($\#_{\mathbf{alt}}\mathbf{X}$). Suppose that $b\#X \in \nabla$ for all $b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ so that $\nabla \vdash a\#\phi\hat{}\pi\cdot X$ is derivable by ($\#_{\mathbf{alt}}\mathbf{X}$). Then, by case (4) of Lemma 2.5.4 we obtain $\nabla \vdash b\#X$ and by the assumptions, $\Delta \vdash b\#X\sigma$. Using the inductive hypothesis on the unpacked definition of $\mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ we obtain $\{b \mid b \in \mathscr{A} \wedge \Delta \nvdash a\#(\phi(\pi(b))\sigma)\}$ such that $\mathtt{fresh}(\Delta, a, (\phi\sigma)\hat{}\pi)$ where $\mathtt{fresh}(\Delta, a, (\phi\sigma)\hat{}\pi) \subseteq \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$. Using ($\#_{\mathbf{alt}}\mathbf{X}$) we derive $\nabla \vdash a\#(\pi\cdot\sigma(X))(\phi\sigma)$ and the result follows by Definition 2.3.9.
- The case ($\#_{\mathbf{alt}}[\mathbf{a}]$). Suppose $\Delta \vdash \nabla\sigma$. Using ($\#_{\mathbf{alt}}[\mathbf{a}]$) we obtain $\nabla \vdash a\#[a]t$. By inductive hypothesis, $\Delta \vdash a\#[a]t\sigma$. The result follows by Definition 2.3.9.
- The case ($\#_{\mathbf{alt}}[\mathbf{b}]$). Suppose $\nabla \vdash a\#t$, $\Delta \vdash \nabla\sigma$. By inductive hypothesis, $\Delta \vdash a\#t\sigma$. Using ($\#_{\mathbf{alt}}[\mathbf{b}]$), $\Delta \vdash a\#[b]t\sigma$. The result follows by application of Definition 2.3.9.
- The case ($\#_{\mathbf{alt}}\mathbf{f}$). Suppose $\nabla \vdash a\#t$, $\Delta \vdash \nabla\sigma$. By inductive hypothesis, $\Delta \vdash a\#t\sigma$. Using ($\#_{\mathbf{alt}}\mathbf{f}$), $\Delta \vdash a\#ft\sigma$. The result follows by Definition 2.3.9.

- The case ($\#_{\mathbf{alt}}\mathbf{tupl}$). Suppose $\nabla \vdash a\#t_1, \ldots, \nabla \vdash a\#t_n$, $\Delta \vdash \nabla\sigma$. By inductive hypothesis, $\Delta \vdash a\#t_1\sigma, \ldots, \Delta \vdash a\#t_n\sigma$. Using ($\#_{\mathbf{alt}}\mathbf{tupl}$), $\Delta \vdash a\#(t_1\sigma, \ldots, t_n\sigma)$. The result follows by Definition 2.3.9.

For the second claim.
By induction on the derivation of $\nabla \vdash s \approx_\alpha t$.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{a}$). By Definition 2.3.9, $a\sigma = a$. The result follows.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$). Suppose $a\#X \in \nabla$ for all $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. By case (4) of Lemma 2.5.4 we have $\nabla \vdash a\#X$ and applying the first claim of this lemma we show that $\Delta \vdash a\#\sigma(X)$. Now we apply the inductive hypothesis on the definition of $\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$: $\{a \mid a \in \mathscr{A} \wedge \Delta \not\vdash (\phi(\pi(a)))\sigma \approx_\alpha (\phi'(\pi'(a)))\sigma\}$ and we are able to construct $\mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$ where $\mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi') \subseteq \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. The result follows by application of ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$) and Definition 2.3.9.

- The case ($\approx\alpha_{\mathbf{alt}}[\mathbf{a}]$). Suppose $\nabla \vdash s \approx_\alpha t$, $\Delta \vdash \nabla\sigma$. Then, by inductive hypothesis, $\Delta \vdash s\sigma \approx_\alpha t\sigma$. Using ($\approx\alpha_{\mathbf{alt}}[\mathbf{a}]$), $\Delta \vdash [a]s\sigma \approx_\alpha [a]t\sigma$. The result follows by application of Definition 2.3.9.

- The case ($\approx\alpha_{\mathbf{alt}}[\mathbf{b}]$). Suppose $\nabla \vdash (a\ b)\cdot s \approx_\alpha t$, $\nabla \vdash b\#s$ and $\Delta \vdash \nabla\sigma$. Using the preceding Lemma, $\Delta \vdash b\#s\sigma$, and by inductive hypothesis, $\Delta \vdash (a\ b)\cdot s\sigma \approx_\alpha t\sigma$. Using ($\approx\alpha_{\mathbf{alt}}[\mathbf{b}]$), $\Delta \vdash [a]s\sigma \approx_\alpha [b]t\sigma$. The result follows by application of Definition 2.3.9.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{f}$). Suppose $\nabla \vdash s \approx_\alpha t$, $\Delta \vdash \nabla\sigma$. Then, by inductive hypothesis, $\Delta \vdash s\sigma \approx_\alpha t\sigma$. Using ($\approx\alpha_{\mathbf{alt}}\mathbf{f}$), $\Delta \vdash fs\sigma \approx_\alpha ft\sigma$. The result follows by application of Definition 2.3.9.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{tupl}$). Suppose $\nabla \vdash s_1 \approx_\alpha t_1, \ldots, \nabla \vdash s_n \approx_\alpha t_n$ and $\Delta \vdash \nabla\sigma$. Then, by inductive hypothesis, $\Delta \vdash s_1\sigma \approx_\alpha t_1\sigma, \ldots, \Delta \vdash s_n\sigma \approx_\alpha t_n\sigma$. Using ($\approx\alpha_{\mathbf{alt}}\mathbf{tupl}$), $\Delta \vdash (s_1\sigma, \ldots, s_n\sigma) \approx_\alpha (t_1\sigma, \ldots, t_n\sigma)$. The result follows by application of Definition 2.3.9. ■

Proof of Lemma 2.5.13.
If $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ *for all* $X \in V(s)$, *then* $\nabla \vdash s\sigma \approx_\alpha s\sigma'$.

Proof. By induction on the syntax of s.

- The case ($s = a$). By Definition 2.1.10 we have $V(a) = \varnothing$ thus there is nothing to prove.

- The case ($s = \phi\hat{}\pi\cdot X$). Suppose $\nabla \vdash Y\sigma \approx_\alpha Y\sigma'$ for all $Y \in V(\mathscr{I}mg(\phi)) \cup \{X\}$. By Definition 2.1.10, $V(\mathscr{I}mg(\phi)) \cup \{X\} = V(\phi\hat{}\pi\cdot X)$ such that $\nabla \vdash Y\sigma \approx_\alpha Y\sigma'$ for all

$Y \in V(\phi\hat{}\pi\cdot X)$. By inductive hypothesis, $\nabla \vdash \phi\sigma\hat{}\pi\cdot X\sigma \approx_\alpha \phi\sigma'\hat{}\pi\cdot X\sigma'$. The result follows by Definition 2.3.9.

- The case $(s = [a]t)$. Suppose $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all $X \in V(t)$. By Definition 2.1.10, $V(t) = V([a]t)$ such that $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all $X \in V([a]t)$. By inductive hypothesis, $\nabla \vdash t\sigma \approx_\alpha t\sigma'$. Using $(\approx_{\alpha_{\mathbf{alt}}}[\mathbf{a}])$ we obtain $\nabla \vdash [a]t\sigma \approx_\alpha [a]t\sigma'$. The result follows by Definition 2.3.9.

- The case $(s = ft)$. Suppose $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all $X \in V(t)$. By Definition 2.1.10, $V(t) = V(ft)$ such that $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all $X \in V(ft)$. By inductive hypothesis, $\nabla \vdash t\sigma \approx_\alpha t\sigma'$. Using $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{f})$ we obtain $\nabla \vdash ft\sigma \approx_\alpha ft\sigma'$. The result follows by Definition 2.3.9.

- The case $(s = (t_1,\ldots,t_n))$. Suppose $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all $X \in \bigcup_{1\leq i\leq n} V(t_i)$. By Definition 2.1.10, $\bigcup_{1\leq i\leq n} V(t_i) = V(t_1,\ldots,t_n)$ such that $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all $X \in V(t_1,\ldots,t_n)$. By inductive hypothesis, $\nabla \vdash t_i\sigma \approx_\alpha t_i\sigma'$ for all $1 \leq i \leq n$. Using $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{tupl})$ we obtain $\nabla \vdash (t_1\sigma,\ldots,t_n\sigma) \approx_\alpha (t_1\sigma',\ldots,t_n\sigma')$. The result follows by Definition 2.3.9.

■

Proof of Lemma 2.5.14.
If $\nabla \vdash n\#s$ *for each* $n \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$, *then* $\nabla \vdash (\pi\cdot s)\phi \approx_\alpha (\pi'\cdot s)\phi'$.

Proof. By induction on the structure of s.

- The case $(s = a)$. Either $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ or not. Suppose $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$, then $a\#a$ contradicts the assumption that ∇ is consistent. Then, it must be the case that $a \notin \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ such that $\nabla \vdash (\pi\cdot a)\phi \approx_\alpha (\pi'\cdot a)\phi'$. Therefore the property holds.

- The case $(s = \phi_1\hat{}\pi_1\cdot X)$. Let $b\#X \in \nabla$ for each $b \in \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$, where $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ such that $\nabla \vdash a\#\phi_1\hat{}\pi_1\cdot X$ is derivable using $(\#_{\mathbf{alt}}\mathbf{X})$. Unpacking the definition of $\mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ we observe that $\nabla \vdash a\#\phi_1(\pi_1(n))$ for all $n \in \mathscr{A} \wedge n \notin \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ and by application of the inductive hypothesis we obtain $\nabla \vdash (\pi\cdot(\phi_1(\pi_1(n))))\phi \approx_\alpha (\pi'\cdot(\phi_1(\pi_1(n))))\phi'$. Hence, $\mathtt{ds}(\nabla, ((\pi\cdot\phi_1)\bullet\phi)\hat{}(\pi\circ\pi_1), ((\pi'\cdot\phi_1)\bullet\phi')\hat{}(\pi'\circ\pi_1)) = \bigcup_{a\in\mathtt{ds}(\nabla,\phi\hat{}\pi,\phi'\hat{}\pi')} \mathtt{fresh}(\nabla, a, \phi_1\hat{}\pi_1)$ and using $(\approx_{\alpha_{\mathbf{alt}}}\mathbf{X})$ we derive $\nabla \vdash ((\pi\cdot\phi_1)\bullet\phi)\hat{}(\pi\circ\pi_1)\cdot X \approx_\alpha ((\pi'\cdot\phi_1)\bullet\phi')\hat{}(\pi'\circ\pi_1)\cdot X$. By Definition 2.3.2 we have, on the LHS, $\nabla \vdash ((\pi\cdot\phi_1)\bullet\phi)\hat{}(\pi\circ\pi_1)\cdot X \approx_\alpha ((\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1)\cdot X)\phi$ and, on the RHS, $\nabla \vdash ((\pi'\cdot\phi_1)\bullet\phi')\hat{}(\pi'\circ\pi_1)\cdot X \approx_\alpha ((\pi'\cdot\phi_1)\hat{}(\pi'\circ\pi_1)\cdot X)\phi'$. By the transitive property from Theorem 2.5.8 we obtain $\nabla \vdash ((\pi\cdot\phi_1)\hat{}(\pi\circ\pi_1)\cdot X)\phi \approx_\alpha ((\pi'\cdot\phi_1)\hat{}(\pi'\circ\pi_1)\cdot X)\phi'$. The result follows by Definition 2.1.8.

- The case ($s = [a]t$). We observe that either $a \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ or not. However, both cases are resolved similarly because of the action of a-substitutions on abstractions which requires choosing distinct representatives on both sides of the equation prior application of a-substitutions. Then, $\mathtt{ds}(\nabla, ((c\ c')\cdot\phi)\hat{}(c\ c')(\pi(a)\ c)\pi, \phi'\hat{}(\pi'(a)\ c')\pi')$ is applied to either case and the proof is solved as follows.

  Suppose $\nabla \vdash c, c'\#t, \mathscr{I}mg(\phi), \mathscr{I}mg(\phi')$. Suppose also that $\nabla \vdash n\#t$ for each $n \in \mathtt{ds}(\nabla, ((c\ c')\cdot\phi^{-c})\hat{}(c\ c')(\pi(a)\ c)\pi, \phi'^{-c'}\hat{}(\pi'(a)\ c')\pi')$. By inductive hypothesis, $\nabla \vdash ((c\ c')(\pi(a)\ c)\pi\cdot t)((c\ c')\cdot\phi^{-c}) \approx_\alpha ((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'}$. Using Definition 2.1.8 we have $((c\ c')(\pi(a)\ c)\pi\cdot t)((c\ c')\cdot\phi^{-c}) = ((c\ c')\cdot((\pi(a)\ c)\pi\cdot t))((c\ c')\cdot\phi^{-c})$. By the first claim of the commutative property from Property 2.5.9,
  $\nabla \vdash ((c\ c')\cdot((\pi(a)\ c)\pi\cdot t))((c\ c')\cdot\phi^{-c}) \approx_\alpha (c\ c')\cdot(((\pi(a)\ c)\pi\cdot t)\phi^{-c})$. Then, by the transitive property from Theorem 2.5.8 we obtain $\nabla \vdash (c\ c')\cdot(((\pi(a)\ c)\pi\cdot t)\phi^{-c}) \approx_\alpha ((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'}$. Using $(\approx\alpha_{\mathbf{alt}}[\mathbf{b}])$ we derive
  $\nabla \vdash [c]((\pi(a)\ c)\pi\cdot t)\phi^{-c}) \approx_\alpha [c']((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'}$. Then, by Definition 2.1.8, $[c]((\pi(a)\ c)\pi\cdot t)\phi) = [c]((\pi(a)\ c)\cdot(\pi\cdot t))\phi)$. Applying Definition 2.3.2 we have, on the LHS, $\nabla \vdash [c]((\pi(a)\ c)\cdot(\pi\cdot t))\phi^{-c}) \approx_\alpha ([\pi(a)]\pi\cdot t)\phi)$, and on the RHS,
  $\nabla \vdash [c']((\pi'(a)\ c')\pi'\cdot t)\phi'^{-c'} \approx_\alpha ([\pi'(a)]\pi'\cdot t)\phi'$. By Definition 2.1.8, $([\pi(a)]\pi\cdot t)\phi) = (\pi\cdot[a]t)\phi)$ on the LHS and $([\pi'(a)]\pi'\cdot t)\phi' = (\pi'\cdot([a]t))\phi'$ on the RHS. The result follows by the transitive property from Theorem 2.5.8.

- The case ($s = ft$). Suppose $\nabla \vdash n\#t$ for each $n \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. By inductive hypothesis, $\nabla \vdash (\pi\cdot t)\phi \approx_\alpha (\pi'\cdot t)\phi'$. Using $(\#_{\mathbf{alt}}\mathbf{f})$, $\nabla \vdash n\#ft$. Applying $(\approx\alpha_{\mathbf{alt}}\mathbf{f})$, $\nabla \vdash f(\pi\cdot t)\phi \approx_\alpha f(\pi'\cdot t)\phi'$. By application of Definition 2.3.2, $\nabla \vdash f(\pi\cdot t)\phi \approx_\alpha (f\pi\cdot t)\phi$ on the LHS and $\nabla \vdash f(\pi'\cdot t)\phi' \approx_\alpha (f\pi'\cdot t)\phi'$ on the RHS. By application of Definition 2.1.8, $(f\pi\cdot t)\phi = (\pi\cdot ft)\phi$ on the LHS and $(f\pi'\cdot t)\phi' = (\pi'\cdot ft)\phi'$ on the RHS. The result follows by the transitive property from Theorem 2.5.8.

- The case ($s = (s_1 \ldots, s_n)$). Suppose $\nabla \vdash n\#s_1 \cdots \nabla \vdash n\#s_n$ for each $n \in ds(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. By inductive hypothesis, $\nabla \vdash (\pi\cdot s_1)\phi \approx_\alpha (\pi'\cdot s_1)\phi', \ldots, \nabla \vdash (\pi\cdot s_n)\phi \approx_\alpha (\pi'\cdot s_n)\phi'$. Using $(\#_{\mathbf{alt}}\mathbf{tupl})$, $\nabla \vdash n\#(s_1, \ldots, s_n)$. Using $(\approx\alpha_{\mathbf{alt}}\mathbf{tupl})$,
  $\nabla \vdash ((\pi\cdot s_1)\phi, \ldots, (\pi\cdot s_n)\phi) \approx_\alpha ((\pi'\cdot s_1)\phi', \ldots, (\pi'\cdot s_n)\phi')$. By application of Definition 2.3.2, $\nabla \vdash ((\pi\cdot s_1)\phi, \ldots, (\pi\cdot s_n)\phi) \approx_\alpha (\pi\cdot s_1, \ldots, \pi\cdot s_n)\phi$ on the LHS and
  $\nabla \vdash ((\pi'\cdot s_1)\phi', \ldots, (\pi'\cdot s_n)\phi') \approx_\alpha (\pi'\cdot s_1, \ldots, \phi'\cdot s_n)\phi'$ on the RHS. By application of Definition 2.1.8 $(\pi\cdot s_1, \ldots, \pi\cdot s_n)\phi = (\pi\cdot(s_1, \ldots, s_n))\phi$ on the LHS and
  $(\pi'\cdot s_1, \ldots, \pi'\cdot s_n)\phi' = \pi'\cdot(s_1, \ldots, s_n)\phi'$ on the RHS. The result follows by the transitive property from Theorem 2.5.8.

■

Proof of Lemma 2.5.15. *If* $\nabla \vdash (\pi{\cdot}s)\phi \approx_\alpha (\pi'{\cdot}s)\phi'$ *then* $\nabla \vdash n\#s$ *for each* $n \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$.

Proof. By induction on the syntax of s.

- The case $(s = a)$. Suppose $\nabla \vdash \phi(\pi(a)) \approx_\alpha \phi'(\pi'(a))$. Then, by definition of notation $\mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$, it is the case that $a \notin \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$ and thus the result follows by $(\#_{\mathbf{alt}}\mathbf{ab})$.

- The case $(s = X)$. Suppose $\nabla \vdash \phi\hat{}\pi{\cdot}X \approx_\alpha \phi'\hat{}\pi'{\cdot}X$. Then, the result follows immediately by using $(\approx\alpha_{\mathbf{alt}}\mathbf{X})$.

- The case $(s = [a]s)$. Suppose $\nabla \vdash ((c\ c')(c\ \pi(a))\pi{\cdot}s)((c\ c'){\cdot}\phi^{-c}) \approx_\alpha ((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'}$ where $\nabla \vdash c, c'\#s, \mathscr{I}mg(\phi), \mathscr{I}mg(\phi')$. Then, by the inductive hypothesis, $\nabla \vdash n\#s$ for each $n \in \mathtt{ds}(\nabla, ((c\ c'){\cdot}\phi^{-c})\hat{}(c\ c')(c\ \pi(a))\pi, \phi'^{-c'}\hat{}(c'\ \pi'(a))\pi')$ where $\mathtt{ds}(\nabla, ((c\ \ c'){\cdot}\phi^{-c})\hat{}(c\ \ c')(c\ \ \pi(a))\pi, \phi'^{-c'}\hat{}(c'\ \ \pi'(a))\pi') = \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi') \setminus \{c, c'\}$. Using $(\#_{\mathbf{alt}}[\mathbf{a}])$ or $(\#_{\mathbf{alt}}[\mathbf{b}])$ we derive $\nabla \vdash n\#[a]s$.

  By the second claim from Property 2.5.9, we have, $\nabla \vdash (((c\ c')(c\ \pi(a))\pi{\cdot}s)((c\ c'){\cdot}\phi^{-c}) \approx_\alpha (c\ c'){\cdot}((c\ \pi(a))\pi{\cdot}s)\phi^{-c'})$. Using the transitive property from Theorem 2.5.8 we obtain $\nabla \vdash (c\ c'){\cdot}(((c\ \pi(a))\pi{\cdot}s)\phi^{-c}) \approx_\alpha ((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'}$. By using $(\approx\alpha_{\mathbf{alt}}[\mathbf{b}])$ we derive $\nabla \vdash [c](((c\ \pi(a))\pi{\cdot}s)\phi^{-c}) \approx_\alpha [c'](((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'}$. By Definition 2.3.2 we have, on the LHS, $\nabla \vdash [c](((c\ \pi(a))\pi{\cdot}s)\phi^{-c}) \approx_\alpha ([\pi(a)]\pi{\cdot}s)\phi$, and on the RHS, $\nabla \vdash [c'](((c'\ \pi'(a))\pi'{\cdot}s)\phi'^{-c'} \approx_\alpha ([\pi'(a)]\pi'{\cdot}s)\phi'$. Applying the transitive property from Theorem 2.5.8, $\nabla \vdash ([\pi(a)]\pi{\cdot}s)\phi) \approx_\alpha ([\pi'(a)]\pi'{\cdot}s)\phi'$. The result follows by Definition 2.1.8.

- The case $(s = ft)$. Suppose $\nabla \vdash (\pi{\cdot}s)\phi \approx_\alpha (\pi'{\cdot}s)\phi'$. Then, by inductive hypothesis, $\nabla \vdash n\#s$ for each $n \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. Using $(\#_{\mathbf{alt}}\mathbf{f})$, $\nabla \vdash n\#fs$. Using $(\approx\alpha_{\mathbf{alt}}\mathbf{f})$, $\nabla \vdash f(\pi{\cdot}s)\phi \approx_\alpha f(\pi'{\cdot}s)\phi'$. By Definition 2.3.2 we have, $\nabla \vdash f(\pi{\cdot}s)\phi \approx_\alpha (f(\pi{\cdot}s))\phi$ on the LHS, and $\nabla \vdash f(\pi'{\cdot}s)\phi' \approx_\alpha (f(\pi'{\cdot}s))\phi'$ on the RHS. By the property of transitivity from Theorem 2.5.8 we obtain $\nabla \vdash (f(\pi{\cdot}s))\phi \approx_\alpha (f(\pi'{\cdot}s))\phi'$. The result follows by Definition 2.1.8.

- The case $(s = (s_1, \ldots, s_n))$. Suppose $\nabla \vdash (\pi{\cdot}s_1)\phi \approx_\alpha (\pi'{\cdot}s_1)\phi' \cdots \nabla \vdash (\pi{\cdot}s_n)\phi \approx_\alpha (\pi'{\cdot}s_n)\phi'$. Then, by inductive hypothesis, $\nabla \vdash n\#s_1 \cdots \nabla \vdash n\#s_n$ for each $n \in \mathtt{ds}(\nabla, \phi\hat{}\pi, \phi'\hat{}\pi')$. Using $(\#_{\mathbf{alt}}\mathbf{tupl})$, $\nabla \vdash n\#(s_1, \ldots, s_n)$. Using $(\approx\alpha_{\mathbf{alt}}\mathbf{tupl})$, $\nabla \vdash ((\pi{\cdot}s_1)\phi, \ldots, (\pi{\cdot}s_n)\phi) \approx_\alpha ((\pi'{\cdot}s_1)\phi', \ldots, (\pi'{\cdot}s_n)\phi')$. By Definition 2.3.2 we have, on the LHS, $\nabla \vdash ((\pi{\cdot}s_1)\phi, \ldots, (\pi{\cdot}s_n)\phi) \approx_\alpha (\pi{\cdot}s_1, \ldots, \pi{\cdot}s_n)\phi$, and on the RHS,

$\nabla \vdash ((\pi'\cdot s_1)\phi', \ldots, (\pi'\cdot s_n)\phi') \approx_\alpha (\pi'\cdot s_1, \ldots, \pi'\cdot s_n)\phi'$, By the property of transitivity from Theorem 2.5.8 we obtain $\nabla \vdash (\pi\cdot s_1, \ldots, \pi\cdot s_n)\phi \approx_\alpha (\pi'\cdot s_1, \ldots, \pi'\cdot s_n)\phi'$. The result follows by Definition 2.1.8.

■

Proof of Lemma 2.5.16.

If $\nabla \vdash n\#s$ *for each* $n \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$*, then* $\nabla \vdash a\#(\pi\cdot s)\phi$.

Proof. By induction on the structure of s.

- The case $(s = b)$. Either $b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ or not. Suppose that $b \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$, then $b\#b$ contradicts the assumption that ∇ is consistent. Therefore it must be the case that $b \notin \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ such that $\nabla \vdash a\#\phi(\pi(b))$. Thus, the property holds.

- The case $(s = \phi'\hat{}\pi'\cdot X)$. Let $b\#X \in \nabla$ for each $b \in \mathtt{fresh}(\nabla, n, \phi'\hat{}\pi')$ and each $n \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ such that $\nabla \vdash n\#\phi'\hat{}\pi'\cdot X$ is derivable using $(\#_{\mathbf{alt}}\mathbf{X})$. Unpacking the definition of $\mathtt{fresh}(\nabla, n, \phi'\hat{}\pi')$ we observe that $\nabla \vdash n\#\phi'(\pi'(c))$ for all $c \in (\mathbb{A} \setminus \mathtt{fresh}(\nabla, n, \phi'\hat{}\pi'))$, and by application of the inductive hypothesis we obtain $\nabla \vdash a\#(\pi\cdot(\phi'(\pi'(c))))\phi$. Hence, $\mathtt{fresh}(\nabla, a, ((\pi\cdot\phi') \bullet \phi)\hat{}(\pi \circ \pi')) = \mathtt{fresh}(\nabla, n, \phi'\hat{}\pi') \cup \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$, and using $(\#_{\mathbf{alt}}\mathbf{X})$ we derive $\nabla \vdash a\#((\pi\cdot\phi') \bullet \phi)\hat{}(\pi \circ \pi')\cdot X$. By Definition 2.3.2 we have $\nabla \vdash a\#((\pi\cdot\phi')\hat{}(\pi \circ \pi')\cdot X)\phi$. The result follows by Definition 2.1.8.

- The case $(s = [a]t)$. Either $a \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ or not. Observe, however, that due to the action of a-substitutions on abstractions, both cases are resolved similarly. Hence, we construct a derivation as follows: Suppose $\nabla \vdash n\#t$ for each $n \in \mathtt{fresh}(\nabla, a, \phi^{-\pi(c)}\hat{}(\pi(a)\ \pi(c))\pi)$ where $\nabla \vdash \pi(c)\#\pi\cdot t, \mathscr{F}mg(\phi)$. Note that, by some basic group theory it is verifiable that $\mathtt{fresh}(\nabla, a, \phi^{-\pi(c)}\hat{}(\pi(a)\ \pi(c))\pi) = \mathtt{fresh}(\nabla, a, \phi^{-\phi(c)}\hat{}\pi) \setminus \{a\}$. Then, by inductive hypothesis it follows that, $\nabla \vdash a\#((\pi(a)\ \pi(c))\pi\cdot t)\phi^{-\pi(c)}$. Using $(\#[\mathbf{b}])$ we obtain $\nabla \vdash a\#[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot t)\phi^{-\pi(c)}$. By Definition 2.1.8, $[\pi(c)]((\pi(a)\ \pi(c))\pi\cdot t)\phi = [\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot t))\phi^{-\pi(c)}$. By Definition 2.3.2, $[\pi(c)]((\pi(a)\ \pi(c))\cdot(\pi\cdot t))\phi^{-\pi(c)} \approx_\alpha ([\pi(a)]\pi\cdot t)\phi$. The result follows by Definition 2.1.8.

- The case $(s = [b]t)$ is similar to the previous case and thus omitted.

- The case $(s = ft)$. Suppose $\nabla \vdash n\#t$ for each $n \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$. By inductive hypothesis it follows that $\nabla \vdash (\pi\cdot t)\phi$. Using $(\#_{\mathbf{alt}}\mathbf{f})$ we have $\nabla \vdash f(\pi\cdot t)\phi$. By Definition 2.3.2, $\nabla \vdash (f(\pi\cdot t))\phi$. The result follows by Definition 2.1.8.

- The case $(s = (s_1, \ldots, s_n))$. Suppose $\nabla \vdash n\#s_1 \cdots \nabla \vdash n\#s_n$ for each $n \in \texttt{fresh}(\nabla, a, \phi\hat{}\pi)$. By inductive hypothesis it follows that $\nabla \vdash (\pi \cdot s_1)\phi \cdots \nabla \vdash (\pi \cdot s_n)\phi$. Using $(\#_{\mathbf{alt}}\mathbf{tupl})$ we have $\nabla \vdash ((\pi \cdot s_1)\phi, \ldots, (\pi \cdot s_n)\phi)$. By Definition 2.3.2, $\nabla \vdash (\pi \cdot s_1, \ldots, \pi \cdot s_n)\phi$. The result follows by Definition 2.1.8.

■

Proof of Lemma 2.5.17.

If $\nabla \vdash a\#(\pi \cdot s)\phi$ *then* $\nabla \vdash n\#s$ *for each* $n \in \texttt{fresh}(\nabla, a, \phi\hat{}\pi)$.

Proof. By induction on the structure of s.

- The case $(s = a)$. Suppose $\nabla \vdash \phi(\pi(a))$. Then, by definition of notation $\texttt{fresh}(\nabla, a, \phi\hat{}\pi)$ it can only be that $a \notin \texttt{fresh}(\nabla, a, \phi\hat{}\pi)$ and thus the result follows by $(\#_{\mathbf{alt}}\mathbf{ab})$.

- The case $(s = X)$. Suppose $\nabla \vdash a\#\phi\hat{}\pi \cdot X$. Then, the result follows immediately by using $(\#_{\mathbf{alt}}\mathbf{X})$.

- The case $(s = [a]t)$. Suppose $\nabla \vdash a\#((c\ \pi(a))\pi \cdot t)\phi^{-c}$ where $\nabla \vdash c\#\pi \cdot t, \mathscr{I}mg(\phi)$. Then, by inductive hypothesis it follows that $\nabla \vdash n\#t$ for each $n \in \texttt{fresh}(\nabla, a, \phi^{-c}\hat{}(c\ \pi(a))\pi)$ where, by group theory, we verify that $\texttt{fresh}(\nabla, a, \phi^{-c}\hat{}(c\ \pi(a))\pi) = \texttt{fresh}(\nabla, a, \phi\hat{}\pi) \setminus \{a\}$. Using $(\#_{\mathbf{alt}}[\mathbf{a}])$ or $(\#_{\mathbf{alt}}[\mathbf{b}])$ we derive $\nabla \vdash n\#[a]t$ and, by application of $(\#_{\mathbf{alt}}[\mathbf{b}])$, we obtain $\nabla \vdash a\#[c]((c\ \pi(a))\pi \cdot t)\phi^{-c}$. By Definition 2.3.2, $\nabla \vdash a\#([\pi(a)]\pi \cdot t)\phi$. The result follows by Definition 2.1.8.

  The case $(s = ft)$. Suppose $\nabla \vdash a\#(\pi \cdot t)\phi$. Then, by inductive hypothesis, $\nabla \vdash n\#t$ for each $n \in \texttt{fresh}(\nabla, a, \phi\hat{}\pi)$. Using $(\#_{\mathbf{alt}}\mathbf{f})$ twice we get, $\nabla \vdash n\#fs$ and $\nabla \vdash a\#f(\pi \cdot t)\phi$. By Definition 2.3.2, $\nabla \vdash a\#(f(\pi \cdot t))\phi$. The result follows by Definition 2.1.8.

- The case $(s = (s_1, \ldots, s_n))$. Suppose $\nabla \vdash a\#(\pi \cdot s_1)\phi \cdots \nabla \vdash a\#(\pi \cdot s_n)\phi$. Then, by inductive hypothesis, $\nabla \vdash n\#s_1 \cdots \nabla \vdash n\#s_n$ for each $n \in \texttt{fresh}(\nabla, a, \phi\hat{}\pi)$. Using $(\#_{\mathbf{alt}}\mathbf{tupl})$ twice we get, $\nabla \vdash n\#(s_1, \ldots, s_n)$ and $\nabla \vdash a\#((\pi \cdot s_1)\phi, \ldots, (\pi \cdot s_1)\phi)$. By Definition 2.3.2, $\nabla \vdash a\#(\pi \cdot s_1, \ldots, \pi \cdot s_n)\phi$. The result follows by Definition 2.1.8.

■

Proof of Lemma 2.5.18.

Let s,t *be any pair of terms and* ∇ *a freshness context such that* $\nabla \vdash s \approx_\alpha t$. *Let* ϕ *be an a-substitution. Then,* $\nabla \vdash s\phi \approx_\alpha t\phi$.

Proof. By induction on the derivation of $\nabla \vdash s \approx_\alpha t$.

- The case $(\approx\alpha_{\mathbf{alt}}\mathbf{a})$. Suppose $\nabla \vdash a \approx_\alpha a$. By inductive hypothesis, $\nabla \vdash \phi(a) \approx_\alpha \phi(a)$ always.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$). Suppose $\nabla \vdash \mathtt{ds}(\nabla, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)\#X$ so that $\nabla \vdash \phi_1\hat{}\pi_1 \cdot X \approx_\alpha \phi_2\hat{}\pi_2 \cdot X$ is derivable using ($\approx\alpha_{\mathbf{alt}}$). By inductive hypothesis on the definition of $\mathtt{ds}(\nabla, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$ we observe that $\mathtt{ds}(\nabla, (\phi_1 \bullet \phi)\hat{}\pi_1, (\phi_2 \bullet \phi)\hat{}\pi_2) \subseteq \mathtt{ds}(\nabla, \phi_1\hat{}\pi_1, \phi_2\hat{}\pi_2)$. The result follows by application of ($\approx\alpha_{\mathbf{alt}}\mathbf{X}$).

- The case ($\approx\alpha_{\mathbf{alt}}[\mathbf{a}]$). Let $\nabla \vdash c, c'\#s, \mathscr{I}mg(\phi)$ and $\nabla \vdash c'\#t, \mathscr{I}mg(\phi')$ also. Assume $\nabla \vdash (c\ c')(a\ c)\cdot s \approx_\alpha (a\ c')\cdot t$. Then, by inductive hypothesis, we have $\nabla \vdash ((c\ c')(a\ c)\cdot s)\phi^{-c} \approx_\alpha ((a\ c')\cdot t)\phi^{-c'}$. Since $\mathtt{ds}(\nabla, \phi^{-c}\hat{}(c\ c')(a\ c), ((c\ c')\cdot\phi^{-c'})\hat{}(c\ c')(a\ c)) = \varnothing$, it is the case that, by Lemma 2.5.14, $\nabla \vdash ((c\ c')(a\ c)\cdot s)\phi^{-c} \approx_\alpha ((c\ c')(a\ c)\cdot s)((c\ c')\cdot\phi^{-c})$. By Definition 2.1.8 we have $((c\ c')(a\ c)\cdot s)((c\ c')\cdot\phi^{-c}) = ((c\ c')\cdot((a\ c)\cdot s))((c\ c')\cdot\phi^{-c})$. By Property 2.5.9, $\nabla \vdash ((c\ c')\cdot((a\ c)\cdot s))((c\ c')\cdot\phi^{-c}) \approx_\alpha (c\ c')\cdot(((a\ c)\cdot s)\phi^{-c})$. Applying the transitive property from Theorem 2.5.8 we obtain $\nabla \vdash (c\ c')\cdot(((a\ c)\cdot s)\phi^{-c}) \approx_\alpha ((a\ c')\cdot t)\phi^{-c'}$. Using ($\approx\alpha_{\mathbf{alt}}[\mathbf{b}]$), $\nabla \vdash [c](((a\ c)\cdot s)\phi^{-c}) \approx_\alpha [c']((a\ c')\cdot t)\phi^{-c'}$. By application of Definition 2.3.2, we have, on the LHS, $\nabla \vdash [c](((a\ c)\cdot s)\phi^{-c}) \approx_\alpha ([a]s)\phi^{-c}$ and, on the RHS, $\nabla \vdash [c']((a\ c')\cdot t)\phi^{-c} \approx_\alpha ([a]t)\phi^{-c}$. The result follows by the transitive property from Theorem 2.5.8.

- The case ($\approx\alpha_{\mathbf{alt}}[\mathbf{b}]$). Let $\nabla \vdash b\#s$, $\nabla \vdash c, c'\#s, \mathscr{I}mg(\phi)$ and $\nabla \vdash c'\#t, \mathscr{I}mg(\phi')$ also. Suppose $\nabla \vdash (c\ c')(a\ c)\cdot s \approx_\alpha (b\ c')t$. Then, by inductive hypothesis, $\nabla \vdash ((c\ c')(a\ c)\cdot s)\phi^{-c} \approx_\alpha ((b\ c')t)\phi^{-c'}$. The result follows similarly to the proof for case ($\approx\alpha_{\mathbf{alt}}[\mathbf{a}]$) and thus omitted from here.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{f}$). Suppose $\nabla \vdash s \approx_\alpha t$, by inductive hypothesis, $\nabla \vdash s\phi \approx_\alpha t\phi$. Using ($\approx\alpha_{\mathbf{alt}}\mathbf{f}$), $\nabla \vdash fs\phi \approx_\alpha ft\phi$. By Definition 2.3.2, $fs\phi = (fs)\phi$. Then, the result follows by the transitive property from Theorem 2.5.8.

- The case ($\approx\alpha_{\mathbf{alt}}\mathbf{tupl}$). Suppose $\nabla \vdash s_1 \approx_\alpha t_1 \cdots \nabla \vdash s_n \approx_\alpha t_n$, by inductive hypothesis, $\nabla \vdash s_1\phi \approx_\alpha t_1\phi, \ldots, \nabla \vdash s_n\phi \approx_\alpha t_n\phi$. Using ($\approx\alpha_{\mathbf{alt}}\mathbf{tupl}$), $\nabla \vdash (s_1\phi, \ldots, s_n\phi) \approx_\alpha (t_1\phi, \ldots, t_n\phi)$. By application of Definition 2.3.2, $(s_1\phi, \ldots, s_n\phi) = (s_1, \ldots, s_n)\phi$. And the result follows by the transitive property from Theorem 2.5.8.

■

Proof of Lemma 2.4.3.

Let ∇ be a freshness context, $a\#t$ a freshness relation between any atom a and term t. Then, $\nabla \vdash a\#t \iff \nabla \vdash_{alt} a\#t$, using the derivation rules from Definitions 2.2.3 & 2.4.1 respectively.

Proof. By induction on the structure of t. All cases are trivial apart from the variable case which we prove below.

For the *only if* case.

Let b range over $\mathscr{D}om(\phi) \cup \{a\}$. Assume that either $\nabla \vdash a\#\phi(b)$ or $\pi^{-1}(b)\#X \in \nabla$ so that $\nabla \vdash a\#\phi\hat{}\pi\cdot X$ is derivable using (#**X**). Then, there are three cases to prove: the case $n \in \mathscr{A} \wedge n \notin (\mathscr{D}om(\phi) \cup \{a\})$ and one case for each of the two freshness assumptions, that is, the case where if $\nabla \vdash a\#\phi(b)$ then $\nabla \vdash a\#\phi(\pi(c))$ for some $\pi(c) \in (\mathscr{D}om(\phi) \cup \{a\})$ such that $\pi(c) = b$, and the case where if $\pi^{-1}(b)\#X \in \nabla$ then $c\#X \in \nabla$ for some $c \in \mathscr{A}$ such that $\pi(c) = b$.

We begin by showing the cases for the pair of freshness assumptions.

1. If $\nabla \vdash a\#\phi(b)$, then there exists an atom c such that $\pi(c) = b$ and by Definition 2.4.1 we obtain $\nabla \vdash_{alt} a\#\phi(\pi(c))$ where $\pi(c) \in (\mathscr{D}om(\phi) \cup \{a\})$ and $c \notin \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$.

2. If $\pi^{-1}(b)\#X \in \nabla$, then there exists an atom c such that $\pi(c) = b$. Now we have two cases to consider, either $\nabla \vdash a\#\phi(b)$ or $\nabla \nvdash a\#\phi(b)$.

   - If it is also the case that $\nabla \vdash a\#\phi(b)$, then $\nabla \vdash_{alt} a\#\phi(\pi(c))$ and, following the previous case, $c \notin \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$.
   - For the case where $\nabla \nvdash a\#\phi(b)$ we observe that, by Definition 2.4.1, it is also the case that $\nabla \nvdash_{alt} a\#\phi(\pi(c))$ where $\pi(c) \in (\mathscr{D}om(\phi) \cup \{a\})$. Therefore $c \in \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$ such that $c\#X \in \nabla$, as expected.

Finally, for the case $n \in \mathscr{A} \wedge n \notin (\mathscr{D}om(\phi) \cup \{a\})$ we have, trivially, $\nabla \vdash a\#\phi(n)$ since $\phi(n) = n$ and, regardless of whether $\pi^{-1}(n)\#X \in \nabla$ or not, we observe that there exists some $\pi(c)$ where $\pi(c) = n$ and $\pi(c) \notin (\mathscr{D}om(\phi) \cup \{a\})$ such that, following Definition 2.4.1, $\nabla \vdash_{alt} a\#\phi(\pi(c))$. Hence $c \notin \mathtt{fresh}(\nabla, a, \phi\hat{}\pi)$.

Then, the result follows by application of rule (#$_{\mathbf{alt}}$**X**).

The other direction is similar and thus omitted. ■

Proof of Lemma 3.2.6.

The relation $\Longrightarrow$ is confluent so that, if there exists a pair of simplification steps $Pr \Longrightarrow^* Pr_1$ and $Pr \Longrightarrow^* Pr_2$ then there also exists Pr_3 such that $Pr_1 \Longrightarrow^* Pr_3$ and $Pr_2 \Longrightarrow^* Pr_3$.

Proof. By Newman's Lemma (Newman, 1942), in order to prove confluence of the reduction system we must show that application of the derivability algorithm terminates, which it does by previous Lemma 3.2.5, and show also that the reduction rules are *locally confluent*. Local confluence is proved by case analysis on the set of reduction rules. We observe that the LHS formula of each reduction rule is of form $s \approx_\alpha tPr$ or form $a\#t, Pr$ where each rule has a structurally distinct constraint of form $s \approx_\alpha t$ or $a\#t$ and Pr is a problem as in Definition 3.1.3 which is common to each rule. Therefore, although there can be more than one path of reduction, the fact that each rule is structurally distinct does not give rise to

interferences, distinct paths occur in distinct subtrees of the logical formula, separated by symbol ‘, ‘. Hence, we are able to deduce that the reduction rules are locally confluent. Then, the result follows. ■

Appendix B

Additional Proofs of Part III

Proof of Lemma 4.1.3.

- $\langle a\#s\rangle_{\mathtt{nf}}$ *is a collection of sets,* $\{\nabla_i \mid i \in I\}$*, where each* ∇_i *is a freshness context or otherwise,* $\langle a\#s\rangle_{\mathtt{nf}} = \{\bot\}$.
- $\langle s \approx_\alpha t\rangle_{\mathtt{nf}}$ *is a collection of sets,* $\{\nabla_i \mid i \in I\}$*, where each* ∇_i *consists of primitive constraints and (possibly none) clashing equalities of form* $\phi\hat{}\pi{\cdot}X \approx_\alpha t$ *or form* $t \approx_\alpha \phi\hat{}\pi{\cdot}X$ *for some* $X \in \mathscr{X}$ *where* t *is any term with a root symbol distinct to* X *or otherwise,* $\langle s \approx_\alpha t\rangle_{\mathtt{nf}} = \{\bot\}$.
- *An immediate consequence is that* $\langle Pr\rangle_{\mathtt{nf}}$ *is also a collection of sets of form* $\langle s \approx_\alpha t\rangle_{\mathtt{nf}}$ *as above or otherwise,* $\langle Pr\rangle_{\mathtt{nf}} = \bot$.

Proof. It follows from Lemma 3.3.8 and the modification to the set of clashing equalities (see Definition 3.2.2). There is a trivial transformation of any derivability problem Pr from set to formula before application of function $\langle\cdot\rangle_{\mathtt{nf}}$ to Pr. Accordingly, the result of such application, $\langle Pr\rangle_{\mathtt{nf}}$, is also implicitly transformed to a collection of sets as follows, if $\langle Pr\rangle_{\mathtt{nf}} = \bigvee_{0\leq i\leq n} Cr_i$ where $\langle\cdot\rangle_{\mathtt{nf}}$ is the function from Lemma 3.3.8 and each C_i a conjunctive clause of reduced consistent constraints and (possibly none) clashing equalities of form $\phi\hat{}\pi{\cdot}X \approx_\alpha t$ or form $t \approx_\alpha \phi\hat{}\pi{\cdot}X$ where $t \neq \phi'\hat{}\pi'{\cdot}X$ for some $X \in \mathscr{X}$ and $1 \leq i \leq n$. Then $\langle Pr\rangle_{\mathtt{nf}} = \{\{Cr_1\},\ldots,\{Cr_n\}\}$ where $\langle\cdot\rangle_{\mathtt{nf}}$ is in this case the updated function as given above. For the particular case $Pr = \varnothing$, we have a direct transformation to $\langle Pr\rangle_{\mathtt{nf}} = \{\varnothing\}$. We remark that the transformation from set to formula and back to set notation is handled externally to the set of reduction rules, that is, no additional rules for relation $\Longrightarrow$ are added for the set transformation. ■

Proof of Lemma 4.1.17.

If $(\mathbf{F},\sigma) \leq (\mathbf{F}',\sigma')$ *then* $(\mathbf{F},\theta \bullet \sigma) \leq (\mathbf{F}',\theta \bullet \sigma')$.

Proof. Suppose for each $\Delta \in \mathbf{F}'$ there is some v-substitution τ and some $\nabla \in \mathbf{F}$ such that $(\mathbf{F},\sigma) \leq (\mathbf{F}',\sigma')$. Applying Definition 4.1.14 we obtain, $\Delta \vdash \sigma \bullet \tau \approx_\alpha \sigma'$ $\Delta \vdash \nabla\tau$. By Lemma 2.5.13, $\Delta \vdash \theta(X)(\sigma \bullet \tau) \approx_\alpha \theta(X)\sigma$ for all $X \in \mathscr{X}$ and $\Delta \in \mathbf{F}'$. By Definition 2.3.12, $\Delta \vdash \theta(\sigma \bullet \tau) \approx_\alpha \theta\sigma$ By definition of v-substitution composition, $\Delta \vdash \theta \bullet (\sigma \bullet \tau) \approx_\alpha \theta \bullet \sigma$. By the associative property, $\Delta \vdash (\theta \bullet \sigma) \bullet \tau \approx_\alpha \theta \bullet \sigma$. The result follows by Definition 4.1.14. ■

Proof of Lemma 4.1.18.

Suppose $\nabla \vdash \sigma \approx_\alpha \sigma'$ *for all* $\nabla \in \mathbf{F}'$. *If* $(\mathbf{F},\theta) \leq (\mathbf{F}',\sigma)$ *then* $(\mathbf{F},\theta) \leq (\mathbf{F}',\sigma')$.

Proof. Suppose for each $\Delta \in \mathbf{F}'$ there is some v-substitution τ and some $\nabla \in \mathbf{F}$ such that $(\mathbf{F},\theta) \leq (\mathbf{F}',\sigma)$. By application of Definition 4.1.14 we have, $\Delta \vdash \theta \bullet \tau \approx_\alpha \sigma$ and $\Delta \vdash \nabla\tau$. By the assumption and the transitive property in Theorem 2.5.8 we have $\Delta \vdash \theta \bullet \tau \approx_\alpha \sigma'$. The result follows by Definition 4.1.14. ■

Proof of Lemma 5.3.1.

For any finite matching problem Pr, each sequence of transformations

$$(Pr,\mathtt{Id}) \overset{Xs}{\Longrightarrow}_{?\approx} \begin{cases} (Pr_{11},\theta_{11}) \overset{Xs}{\Longrightarrow}_{?\approx} \ldots \overset{Xs}{\Longrightarrow}_{?\approx} (Pr_{1k},\theta_{1k}) \Longrightarrow_{\#} (Pr_{1(k+1)},\theta_{1k}) \Longrightarrow_{\#} \ldots \\ \quad \vdots \\ (Pr_{n1},\theta_{n1}) \overset{Xs}{\Longrightarrow}_{?\approx} \ldots \overset{Xs}{\Longrightarrow}_{?\approx} (Pr_{nm},\theta_{nm}) \Longrightarrow_{\#} (Pr_{n(m+1)},\theta_{nm}) \Longrightarrow_{\#} \ldots \end{cases}$$

terminates, either with $(\{\bot\},\theta)$ *and the pair is discarded from the set of solutions, or with* $(\mathbf{F},\theta)$ *where* $\mathbf{F}$ *is a collection of freshness contexts and* θ *a composition of v-substitutions. Above,* $Xs = V_{RHS}(Pr)$.

Proof. Any unification or freshness constraint fits into one of the cases mentioned on the left-hand side of rules $\Longrightarrow_{?\approx}$ or rules $\Longrightarrow_{\#}$. The set of freshness rules applied during the second phase of the unification algorithm terminates, as proved in Lemma 3.2.5 and updated for the failure rule $(\#\bot)$ in Lemma 3.3.8. Applying the same Lemma 3.2.5 we observe that, during the first phase of the algorithm, the subset of non-instantiating rules in relation $\Longrightarrow_{?\approx}$ also terminates. The termination proof was also updated in Lemma 3.3.8 to show that the property of termination is preserved after addition of failure rule $(\approx_\alpha\bot)$ to the set of α-equality rules. Now, rule $(?\approx\bot)$ is equivalent to rule $(\approx_\alpha\bot)$ except that constraints of form $\phi\hat{}\pi\cdot X \;?\approx_? t$ do not belong to the (updated) set of clashing equalities (see Definition 5.2.5) whenever $X \notin Xs$. Thus, rule $(?\approx)$ preserves termination. Hence, the interesting cases to show are the subset of instantiating rules for $\Longrightarrow_{?\approx}$, namely, rules $(?\approx\textbf{Inst})$ and $(?\approx\textbf{XY})$.

We show that application of instantiating rules reduces the matching problem as follows. For every unification constraint $s \,?{\approx}?\, t \in Pr$, we define a complexity measure $\langle n_1, n_2, n_3 \rangle$ to be the well-founded lexicographically ordered trio of natural numbers n_1, n_2, n_3 where

- n_1 is the number of distinct variables in (s,t),
- n_2 is the size of $s \,?{\approx}?\, t$, that is $|s| + |t|$ (see Definition 2.1.13),
- n_3 is the weight of the predicate, we assign 2 to predicate $?{\approx}?$ and 1 to predicate #.

Now, suppose without loss of generality that $(\{s \,?{\approx}?\, t\}, \mathtt{Id}) \overset{Xs}{\Longrightarrow}_{?\approx} \Pi$ via some instantiating rule. Then, each rule reduces the complexity of $s \,?{\approx}?\, t$ as follows:

($?{\approx}?$**Inst**) Suppose that $s = \phi\hat{}\pi{\cdot}X$ where $X \notin Xs$ and $t \neq \phi\hat{}\pi{\cdot}Y$ for some $Y \in \mathscr{X}$. Then, by application of rule ($?{\approx}$**Inst**) we observe that, $\Pi = \bigcup_{u \in \mathtt{Cap}(t, \mathscr{D}om(\phi))} (\{u(\phi[X \mapsto \pi^{-1}{\cdot}u]) \,?{\approx}?\, t\}, \mathtt{Id} \bullet [X \mapsto \pi^{-1}{\cdot}u])$ and we must prove that Π is a finite set and also that each instance of $\phi\hat{}\pi{\cdot}X \,?{\approx}?\, t$ under $[X \mapsto \pi^{-1}{\cdot}u]$, that is, $u(\phi[X \mapsto \pi^{-1}{\cdot}u]) \,?{\approx}?\, t$, decreases the complexity measure.

Finiteness of Π is proved by verifying that `Cap` terminates. This is the case since the number of recursive calls in $\mathtt{Cap}(t, \mathscr{D}om(\phi))$ is controlled by the set of positions in argument t, which is a finite term, and the size of $\mathscr{D}om(\phi)$ which is also finite. Now, by definition of matching problem (see Definition 5.1.1), for any term $u \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$, it is a fact that $X \notin V(u)$. Therefore, $\{u(\phi[X \mapsto \pi^{-1}{\cdot}u]) \,?{\approx}?\, t\}$ decreases the number of distinct variables by 1 with respect to $\phi\hat{}\pi{\cdot}X \,?{\approx}?\, t$.

Hence rule ($?{\approx}$**Inst**) reduces the matching problem.

($?{\approx}$**XY**) Suppose that $s = \phi\hat{}\pi{\cdot}X$, $t = \phi'\hat{}\pi'{\cdot}Y$ with $X \neq Y$ where $X \notin Xs$. Then, by application of ($?{\approx}$**XY**) we observe that

$$\Pi = \bigcup_{s \in (\mathtt{Cap}(\phi'\hat{}\pi'{\cdot}Y, \mathscr{D}om(\phi)) \cup \{\pi'{\cdot}Y\})} (\{s(\phi[X \mapsto \pi^{-1}{\cdot}s]) \,?{\approx}?\, \phi'\hat{}\pi'{\cdot}Y\}, \mathtt{Id} \bullet [X \mapsto \pi^{-1}{\cdot}s])$$

Using the proof from the previous case, ($?{\approx}$**Inst**), it is easy to deduce that the equation above consist of finite sets where each instance of $\phi\hat{}\pi{\cdot}X \,?{\approx}?\, \phi'\hat{}\pi'{\cdot}Y$ decreases the complexity measure.

Hence rule ($?{\approx}$**XY**) reduces the matching problem.

Finally, we conclude by stating that the property of termination holds for the set of unification and freshness rules described in the matching algorithm (see Definition 5.2.6) such that, for any matching problem Pr, application of the set of rules $\overset{Xs}{\Longrightarrow}_{?\approx}$ and rules $\Longrightarrow_{\#}$ to Pr generates a finite number of distinct paths of reduction. By application of the two-phase

strategy described in Definition 5.2.6, each sequence of transformations resolves to, either $(\{\bot\}, \theta)$ and the pair is discarded from the set of solutions, or with $(\mathbf{F}, \theta)$ where $\nabla \in \mathbf{F}$ is a freshness context and θ a composition of v-substitutions. ■

Proof of Lemma 5.3.2.

The relation $\Longrightarrow$ is confluent such that, if there exists a pair of reduction steps $(Pr, \theta) \Longrightarrow_* (Pr_1, \theta_1)$ and $(Pr, \theta) \Longrightarrow_* (Pr_2, \theta_2)$, then there also exists (Pr_3, θ_3) such that $(Pr_1, \theta_1) \Longrightarrow_* (Pr_3, \theta_3)$ and $(Pr_2, \theta_2) \Longrightarrow_* (Pr_3, \theta_3)$.

Proof. It was already demonstrated in the Confluence proof from Lemma 3.2.6 that relation $\Longrightarrow_{\#}$ is confluent for the set of freshness rules. Moreover, since freshness rules do not share predicate symbols with matching rules, they are also confluent with respect to the matching system described in Definition 5.2.6. Then, we just need to show that matching rules are also confluent. This is the case since matching rules do not interfere with each other under the strategy imposed where rule $({}_?\approx\equiv)$ has the highest priority. Observe that the two instantiating rules which could lead to multiple redexes are uniquely determined by the imposed side conditions. Therefore, using Newman's Lemma (Newman, 1942) and the termination proof from Lemma 5.3.1, we conclude that the matching algorithm is indeed confluent. ■

Proof of Lemma 5.3.7.

- $\langle\{a\#s\}\rangle_{sol} = [\mathtt{Match}(\{a\#s\}, \varnothing)] = \{(\langle\{a\#s\}\rangle_{\mathtt{nf}}, \mathtt{Id})\}$ *if* $\langle\{a\#s\}\rangle_{\mathtt{nf}} \neq \bot$, *where* $\langle\{a\#s\}\rangle_{\mathtt{nf}}$ *is a collection of freshness contexts or otherwise,* $\langle a\#s\rangle_{sol} = \varnothing$;
- $\langle\{s \;{}_?\!\approx_?\; t\}\rangle_{sol} = [\mathtt{Match}(\{s \;{}_?\!\approx_?\; t\}, V_{RHS}(\{s \;{}_?\!\approx_?\; t\}))] = \{(\mathbf{F}_i, \theta_i) \mid i \in I\}$ *where each* $\mathbf{F}_i$ *is a collection of freshness contexts and each* θ_i *is a distinct unifier or otherwise,* $\langle\{s \;{}_?\!\approx_?\; t\}\rangle_{sol} = \varnothing$;
- *An immediate consequence is that* $\langle Pr\rangle_{sol}$ *has also form* $\{(\mathbf{F}_i, \theta_i) \mid i \in I\}$ *as above, or otherwise* $\varnothing$, *if* $\langle C\rangle_{sol} = \varnothing$ *for some* $C \in Pr$.

Proof. The first claim follows by Lemma 4.1.3 for the normal form of freshness constraints where application of Definition 5.3.4 is trivial.

For the second claim, it was proved in Lemma 5.3.1 that the application of both non-instantiating and instantiating rules to non-reduced unification constraints and the application of freshness rules to non-reduced freshness constraints terminates, preserving the structure of a normal form and thus resulting in either a set of solutions $\{(\mathbf{F}_i, \theta_i) \mid i \in \mathrm{I}\}$ where each $\mathbf{F}_i$ is a collection of freshness contexts and each θ_i is a distinct unifier or otherwise, $\varnothing$. Further, by both the definition of matching problem (see Definition 5.1.1) and instantiating rules

generating only idempotent v-substitutions (see Definition 5.2.6), it is a fact that each solution in the set contains only idempotent unifiers and by Definition 5.3.4 idempotent unifiers in the solution are pairwise distinct.

The last bullet point is a corollary of the other two. ■

Proof of Lemma 5.3.8.

Given a matching problem Pr, assume $\Delta \vdash X\sigma \approx_\alpha X\sigma'$ *for all* $X \in V(Pr)$ *for all* $\Delta \in \mathbf{F}$. *Then,* $(\mathbf{F},\sigma) \in \mathscr{U}(Pr)$ *if and only if* $(\mathbf{F},\sigma') \in \mathscr{U}(Pr)$.

Proof. By Definition 4.1.8 it suffices to show $\Delta \vdash s\sigma \approx_\alpha t\sigma$ if and only if $\Delta \vdash s\sigma' \approx_\alpha t\sigma'$ and also $\Delta \vdash a\#r\sigma$ if and only if $\Delta \vdash a\#r\sigma'$ for each constraint of form $a\#r$ or form $s \,{}_?{\approx}_?\, t$ in Pr and each freshness context $\Delta \in \mathbf{F}$.

• For the first case.

Suppose $(\mathbf{F},\sigma) \in \mathscr{U}(Pr)$. By definition of solution to a matching problem as in Definition 5.1.1, $\Delta \vdash s_i\sigma \approx_\alpha t_i\sigma$ for each $s_i \,{}_?{\approx}_?\, t_i \in Pr$ and each $\Delta \in \mathbf{F}$ where $\mathscr{D}om(\sigma) \subseteq (\bigcup_i V(s_i))$ and thus $\Delta \vdash t_i\sigma \approx_\alpha t$. By Definition 3.1.4 we have $V(s) \subseteq V(Pr)$ and $V(t) \subseteq V(Pr)$. By Lemma 2.5.13 we have $\Delta \vdash s\sigma \approx_\alpha s\sigma'$ and $\Delta \vdash t\sigma \approx_\alpha t\sigma'$. By the transitive property from Theorem 2.5.8, it is also the case that $\Delta \vdash t\sigma \approx_\alpha t\sigma' \approx_\alpha t$. Applying the transitive property again, $\Delta \vdash s\sigma' \approx_\alpha t\sigma'$. The result follows by definition of solution to a matching problem as in Definition 5.1.1. The other direction is similar and thus omitted.

• For the second case.

Suppose $(\mathbf{F},\sigma) \in \mathscr{U}(Pr)$. By by definition of solution to a matching problem as in Definition 5.1.1, $\Delta \vdash a\#r\sigma$ for each $a\#r \in Pr$ and each $\Delta \in \mathbf{F}$. By Definition 3.1.4 we have $V(r) \subseteq V(Pr)$. By Lemma 2.5.13 we have $\Delta \vdash r\sigma \approx_\alpha r\sigma'$. By Lemma 2.5.7, $\Delta \vdash a\#r\sigma'$. The result follows by by definition of solution to a matching problem as in Definition 5.1.1. The other direction is similar and thus omitted. ■

Proof of Lemma 5.3.9.

Given a set of matching problems $\Psi(\phi\hat{}\pi, \phi'\hat{}\pi', \varnothing)^A_X$ *(see Definition 5.2.1), a set of atoms* $\mathtt{ds}(\nabla, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$ *(see Equation 2.4), some freshness context* ∇ *and v-substitution* σ *such that* σ *satisfies* ∇ *(see Property 4.1.7), then* $\nabla \vdash a\#\sigma(X)$ *for each* $a \in \mathtt{ds}(\nabla, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi') \iff \nabla \vdash Pr'\sigma$ *for some* $Pr \in \Psi(\phi\hat{}\pi, \phi'\hat{}\pi', \varnothing)^A_X$ *where* Pr' *is* Pr *with constraints of form* $s \,{}_?{\approx}_?\, t \in Pr$ *replaced by constraints of form* $s \approx_\alpha t$ *in* Pr'.

Proof. • For the *if* direction. By induction on the atoms in A where, by Definition 5.2.1, $A = \mathscr{D}omP(\phi,\phi') \cup \mathscr{S}upportP(\pi,\pi')$.

Suppose $A = \varnothing$. By Definition 5.2.1 we have $\Psi(\phi\hat{}\pi, \phi'\hat{}\pi', \varnothing)_X^{\varnothing} = \varnothing$ and then $\nabla \vdash \varnothing\sigma$ always. Following Equation 2.4 it is the case that $\mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi') = \varnothing$. The result follows.

Suppose $a \in A$. By Definition 5.2.1 we have,
$\Psi(\phi\hat{}\pi, \phi'\hat{}\pi', Pr'')_X^A = \Psi(\phi\hat{}\pi, \phi'\hat{}\pi', Pr'' \cup \{a\#X\})_X^{A\setminus\{a\}}$
$\cup\Psi(\phi\hat{}\pi, \phi'\hat{}\pi', Pr'' \cup \{\phi(\pi(a)) \;?\!\approx\!?\; \phi'(\pi'(a))\})_X^{A\setminus\{a\}}$ and therefore we have two cases to show.

1. Suppose $\nabla \vdash Pr'\sigma$ for some $Pr \in \Psi(\phi\hat{}\pi, \phi'\hat{}\pi', Pr'' \cup \{a\#X\})_X^{A\setminus\{a\}}$ where Pr' is Pr with constraints of form $s \;?\!\approx\!?\; t \in Pr$ replaced by constraints $s \approx_\alpha t$ in Pr'. Then, there are two cases to consider depending on whether $\nabla \vdash (\phi(\pi(a)))\sigma \approx_\alpha (\phi'(\pi'(a)))\sigma$ or $\nabla \nvdash (\phi(\pi(a)))\sigma \approx_\alpha (\phi'(\pi'(a)))\sigma$.
   (a) Suppose $\nabla \vdash (\phi(\pi(a)))\sigma \approx_\alpha (\phi'(\pi'(a)))\sigma$. Following Equation 2.4 it is not the case that $a \in \mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$ and therefore there is nothing to prove for this case.
   (b) Suppose $\nabla \nvdash (\phi(\pi(a)))\sigma \approx_\alpha (\phi'(\pi'(a)))\sigma$. Following Equation 2.4 it is the case that $a \in \mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$ and it is a fact that $a\#\sigma(X) \subset Pr\sigma$. Therefore $\nabla \vdash a\#\sigma(X)$ as expected.
2. Suppose $\nabla \vdash Pr'\sigma$ for some
   $Pr \in \Psi(\phi\hat{}\pi, \phi'\hat{}\pi', Pr'' \cup \{\phi(\pi(a)) \;?\!\approx\!?\; \phi'(\pi'(a))\})_X^{A\setminus\{a\}}$ where Pr' is Pr with constraints of form $s \;?\!\approx\!?\; t \in Pr$ replaced by constraints $s \approx_\alpha t$ in Pr'. Then, by Definition 4.1.8 we have $\nabla \vdash (\phi(\pi(a)))\sigma \approx_\alpha (\phi'(\pi'(a)))\sigma$ and following Equation 2.4 it is not the case that $a \in \mathtt{ds}(\Delta, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$ and therefore there is nothing to prove for this case.

The result follows by inductive hypothesis.

- For the *only if* direction. By induction on the atoms in $\mathscr{D}omP(\phi, \phi', \cup)\mathscr{S}upportP(\pi, \pi',)$ and the unpacked definition of $\mathtt{ds}(\nabla, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$.

  The result follows similarly to the previous claim and thus omitted.

■

Proof of Lemma 5.3.10.

Given the set of formulæ $\mathbf{S}nf(\mathtt{DNF}(\bigwedge_{b\in(\mathscr{D}om(\phi)\cup\{a\})} \langle \pi^{-1}(b), a\#\phi(b), X\rangle))$ *(see Definition 3.2.9 for function* $\mathbf{S}nf$ *and Definition 3.3.4 for function* $\langle\cdot,\cdot,\cdot\rangle$*), the set of atoms* $\mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi)$ *(see Equation 2.2), some freshness context* ∇ *and v-substitution* σ *such*

that σ *satisfies* ∇ *(see Property 4.1.7), then* $\nabla \vdash n\#\sigma(X)$ *for each* $n \in \mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi)$ $\iff \nabla \vdash \Delta\sigma$ *for some* $\Delta \in \mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle))$.

Proof. • The *if* direction. By induction on the atoms in $(\mathscr{D}om(\phi) \cup \{a\})$.

Suppose $(\mathscr{D}om(\phi) \cup \{a\}) = \varnothing$. Then, $\mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle)) = \varnothing$ and $\nabla \vdash \varnothing\sigma$ always. Further, it is also the case that $\mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi) = \varnothing$ and thus the result follows.

Suppose $b \in (\mathscr{D}om(\phi) \cup \{a\})$. Unpacking the definition of $\langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle$ we observe there are three cases to examine, the case where $\langle a\#\phi(b)\rangle_{\mathtt{nf}} = \top$, the case where $\langle a\#\phi(b)\rangle_{\mathtt{nf}} = \bot$ and the case where $\langle a\#\phi(b)\rangle_{\mathtt{nf}} = \Gamma$, with Γ a conjunctive clause of consistent freshness constraints. In all three cases, $\langle\cdot\rangle_{\mathtt{nf}}$ is the function defined in Lemma 3.3.8.

1. If $\langle a\#\phi(b)\rangle_{\mathtt{nf}} = \top$, then $\langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle = \top$ and $\nabla \vdash \top\sigma$ always. Further, unpacking the definition of $\mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi)$ it follows that $\nabla \vdash a\#\phi(b)$ and therefore $\pi^{-1}(b) \notin \mathtt{fresh}(\nabla, (\phi\sigma)\hat{}\pi, (\phi'\sigma)\hat{}\pi')$ and there is nothing to prove for this case.
2. If $\langle a\#\phi(b)\rangle_{\mathtt{nf}} = \bot$, then $\langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle = \pi^{-1}(b)\#X$ and, by application of $\mathtt{DNF}$, $\pi^{-1}(b)\#X \in \mathbf{S}(\Delta)$ for all $\Delta \in \mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle))$.
   Now, suppose $\nabla \vdash \Delta\sigma$ for some $\Delta \in \mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle))$.
   Then, unpacking the definition of $\mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi)$, it follows that $\nabla \nvdash a\#(\phi(b))\sigma$ and therefore $\pi^{-1}(b) \in \mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi)$ and the property holds for this case.
3. If $\langle a\#\phi(b)\rangle_{\mathtt{nf}} = \Gamma$ where Γ a conjunctive clause of consistent freshness constraints, then $\langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle = \Gamma \vee \pi^{-1}(b)\#X$ and, by application of $\mathtt{DNF}$, there are two cases to show, either $\Gamma \in \mathbf{S}(\Delta)$ or $\pi^{-1}(b)\#X \in \mathbf{S}(\Delta)$ for some $\Delta \in \mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle))$.
   (a) For the case where $\pi^{-1}(b)\#X \in \mathbf{S}(\Delta)$, the proof follows similarly to case 2.
   (b) For the case where $\Gamma \in \mathbf{S}(\Delta)$, suppose $\nabla \vdash \Delta\sigma$. Then, unpacking the definition of $\mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi)$, it follows that $\nabla \vdash a\#(\phi(b))\sigma$ and therefore $\pi^{-1}(b) \notin \mathtt{fresh}(\nabla, a, (\phi\sigma)\hat{}\pi)$ and there is nothing to prove for this case.

The result follows by inductive hypothesis.

■

Proof of Lemma 5.3.11.

Let Pr_0 be a matching problem and consider a step $(Pr,\theta) \Longrightarrow_{?^{\approx}} \{(Pr'_i,\theta) | i \in I\}$ in its reduction to normal form $\langle Pr_0 \rangle_{\mathtt{nf}_{?^{\approx}}}$ by application of some non-instantiating rule, then $\mathscr{U}(Pr) = \bigcup_{i\in I} \mathscr{U}(Pr'_i)$.

Proof. By case analysis on the non-instantiating rules for the relation $\Longrightarrow_{?^{\approx}}$. For the sake of simplicity, suppose $Pr = \{C\}$, that is, Pr contains only one problem. Since a solution to a matching problem is also a solution to a unification problem, we apply directly Definition 4.1.8 whenever required without loss of generality (recall that $\mathscr{D}om(\theta) \cap V_{RHS}(Pr_0) = \varnothing$)

- The case $(?\approx?\equiv)$. Suppose $(\{s \stackrel{?}{\approx}_? s\},\theta) \stackrel{(?\approx\equiv)}{\Longrightarrow} (\varnothing,\theta)$.

  Suppose also that $(\mathbf{F},\sigma) \in \mathscr{U}(\varnothing)$. By Definition 4.1.8, $\forall \nabla \in \mathbf{F}.\nabla \vdash \varnothing\sigma$ where $\varnothing\sigma = \varnothing$ by Definition 2.3.12. By Reflexivity from Theorem 2.5.8 we have, $\nabla \vdash s \approx_\alpha s$. Using Lemma 2.5.12, $\nabla \vdash s\sigma \approx_\alpha s\sigma$. The result follows by application of Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F},\sigma) \in \mathscr{U}(s \stackrel{?}{\approx}_? s)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash s\sigma \approx_\alpha s\sigma$. Trivially, $s\sigma = s\sigma$ always such that $\nabla \vdash \varnothing$ and by Definition 2.3.12 $\varnothing\sigma = \varnothing$. The result follows by Definition 4.1.8.

- The case $(?\approx[\mathbf{a}])$. Suppose $(\{[a]s \stackrel{?}{\approx}_? [a]t\},\theta) \stackrel{(?\approx[\mathbf{a}])}{\Longrightarrow} (\{s \stackrel{?}{\approx}_? t\},\theta)$.

  Suppose also that $(\mathbf{F},\sigma) \in \mathscr{U}(s \stackrel{?}{\approx}_? t)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash s\sigma \approx_\alpha t\sigma$. Using $(\approx_\alpha[\mathbf{a}])$, $\nabla \vdash [a]s\sigma \approx_\alpha [a]t\sigma$. Applying Definition 2.3.9 we have $\nabla \vdash ([a]s)\sigma \approx_\alpha ([a]t)\sigma$. The result follows by Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F},\sigma) \in \mathscr{U}([a]s \stackrel{?}{\approx}_? [a]t)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash ([a]s)\sigma \approx_\alpha ([a]t)\sigma$. Applying Definition 2.3.9 we have $\nabla \vdash [a]s\sigma \approx_\alpha [a]t\sigma$. Using $(\approx_\alpha[\mathbf{a}])$, $\nabla \vdash s\sigma \approx_\alpha t\sigma$. The result follows by application of Definition 4.1.8.

- The case $(?\approx[\mathbf{b}])$. Suppose $(\{[a]s \stackrel{?}{\approx}_? [b]t\},\theta) \stackrel{(?\approx[\mathbf{b}])}{\Longrightarrow} (\{(a\ b)\cdot s \stackrel{?}{\approx}_? t, b\#s\},\theta)$.

  Suppose also that $(\mathbf{F},\sigma) \in \mathscr{U}((a\ b)\cdot s \stackrel{?}{\approx}_? t, b\#s)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash ((a\ b)\cdot s)\sigma \approx_\alpha t\sigma, b\#s\sigma$. Applying Definition 2.1.8, $\nabla \vdash ((a\ b)\cdot s)\sigma \approx_\alpha (a\ b)\cdot s\sigma$. By the transitive property from Theorem 2.5.8, $\nabla \vdash (a\ b)\cdot s\sigma \approx_\alpha t\sigma, b\#s\sigma$. Using $(\approx_\alpha[\mathbf{b}])$ we derive $\nabla \vdash [a]s\sigma \approx_\alpha [b]t\sigma$. Applying Definition 2.3.9 we have $\nabla \vdash ([a]s)\sigma \approx_\alpha ([b]t)\sigma$. The result follows by Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F},\sigma) \in \mathscr{U}([a]s \stackrel{?}{\approx}_? [b]t)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash ([a]s)\sigma \approx_\alpha ([b]t)\sigma$. Applying Definition 2.3.9 we have $\nabla \vdash [a]s\sigma \approx_\alpha [b]t\sigma$. Using $(\approx_\alpha[\mathbf{b}])$, $\nabla \vdash (a\ b)\cdot s\sigma \approx_\alpha t\sigma, b\#s\sigma$. By Definition 4.1.8, $\nabla \vdash (a\ b)\cdot s\sigma \approx_\alpha ((a\ b)\cdot s)\sigma$. By the transitive property from Theorem 2.5.8 we have, $\nabla \vdash ((a\ b)\cdot s)\sigma \approx_\alpha t\sigma, b\#s\sigma$. The result follows by use of Definition 4.1.8.

- The case $(?\approx\mathbf{f})$. Suppose $(\{fs\ ?\approx?\ ft\},\theta) \overset{(?\approx\mathbf{f})}{\Longrightarrow} (\{s\ ?\approx?\ t\},\theta)$.

  Suppose also that $(\mathbf{F},\sigma) \in \mathscr{U}(s\ ?\approx?\ t)$. By Definition 4.1.8 we have $\forall\nabla \in \mathbf{F}.\nabla \vdash s\sigma \approx_\alpha t\sigma$. Using $(\approx_\alpha\mathbf{f})$, $\nabla \vdash fs\sigma \approx_\alpha ft\sigma$. Applying Definition 2.3.9 we have $\nabla \vdash (fs)\sigma \approx_\alpha (ft)\sigma$. The result follows by Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F},\sigma) \in \mathscr{U}(fs\ ?\approx?\ ft)$. By Definition 4.1.8 we have $\forall\nabla \in \mathbf{F}.\nabla \vdash (fs)\sigma \approx_\alpha (ft)\sigma$. Applying Definition 2.3.9 we have $\nabla \vdash fs\sigma \approx_\alpha ft\sigma$. Using $(\approx_\alpha\mathbf{f})$, $\nabla \vdash s\sigma \approx_\alpha t\sigma$. The result follows by use of Definition 4.1.8.

- The case $(?\approx\mathbf{tupl})$. Suppose $((s_1,\ldots,s_n)\ ?\approx?\ (t_1,\ldots,t_n)) \overset{(?\approx\mathbf{tupl})}{\Longrightarrow} (\{s_1\ ?\approx?\ t_1,\ldots,s_n\ ?\approx?\ t_n\},\theta)$.

  Suppose also that $(\mathbf{F},\sigma) \in \mathscr{U}(s_1\ ?\approx?\ t_1,\ldots,s_n\ ?\approx?\ t_n)$. By Definition 4.1.8 we have, $\forall\nabla \in \mathbf{F}.\nabla \vdash s_1\sigma \approx_\alpha t_1\sigma,\ldots,s_n\sigma \approx_\alpha t_n\sigma$. Using $(\approx_\alpha\mathbf{tupl})$, $\nabla \vdash (s_1\sigma,\ldots,s_n\sigma) \approx_\alpha (t_1\sigma,\ldots,t_n\sigma)$. Applying Definition 2.3.9 we have $\nabla \vdash (s_1,\ldots,s_n)\sigma \approx_\alpha (t_1,\ldots,t_n)\sigma$. The result follows by Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F},\sigma) \in \mathscr{U}((s_1,\ldots,s_n)\ ?\approx?\ (t_1,\ldots,t_n))$. By Definition 4.1.8 we have $\forall\nabla \in \mathbf{F}.\nabla \vdash (s_1,\ldots,s_n)\sigma \approx_\alpha (t_1,\ldots,t_n)\sigma$. Applying Definition 2.3.9 we have, $\nabla \vdash (s_1\sigma,\ldots,s_n\sigma) \approx_\alpha (t_1\sigma,\ldots,t_n\sigma)$. Using $(\approx_\alpha\mathbf{tupl})$, $\nabla \vdash s_1\sigma \approx_\alpha t_1\sigma,\ldots,s_n\sigma \approx_\alpha t_n\sigma$. The result follows by Definition 4.1.8.

- The case $(?\approx\mathbf{X})$. Suppose $(\{\phi\hat{}\pi\cdot X\ ?\approx?\ \phi'\hat{}\pi'\cdot X\},\theta) \overset{(?\approx\mathbf{X})}{\Longrightarrow} \bigcup_{Pr'_i\in\Psi(\phi\hat{}\pi,\phi\hat{}\pi,\varnothing)^A_X} \{(Pr'_i,\theta)\}$.

  Suppose also that $(\mathbf{F},\sigma_i) \in \mathscr{U}(Pr'_i)$ for each $Pr'_i \in \Psi(\phi\hat{}\pi,\phi\hat{}\pi,\varnothing)^A_X$. By Definition 4.1.8 we have, $\forall\nabla \in \mathbf{F}.\nabla \vdash Pr_i^{\approx}\sigma_i$ where $Pr_i^{\approx}$ is Pr'_i with constraints of form $s\ ?\approx?\ t$ replaced by constraints of form $s \approx_\alpha t$. Using Lemma 5.3.9 we obtain $\nabla \vdash a\#\sigma_i(X)$ for each $a \in \mathtt{ds}(\nabla,(\phi\sigma_i)\hat{}\pi,(\phi'\sigma_i)\hat{}\pi')$ such that, applying Lemma 2.5.14 we are able to obtain $\nabla \vdash ((\pi\cdot\sigma_i(X))\phi\sigma_i) \approx_\alpha ((\pi'\cdot\sigma_i(X))\phi'\sigma_i)$. Applying Definition 2.3.9 on both sides of $\approx_\alpha$ we obtain, $\nabla \vdash (\phi\hat{}\pi\cdot X)\sigma_i \approx_\alpha (\phi'\hat{}\pi'\cdot X)\sigma_i$. The result follows by Definition 4.1.8 and the union of sets.

  Otherwise, suppose $(\mathbf{F},\sigma) \in \mathscr{U}(\phi\hat{}\pi\cdot X\ ?\approx?\ \phi\hat{}\pi\cdot X)$. By Definition 4.1.8 we obtain, $\forall\nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi\cdot X)\sigma \approx_\alpha (\phi'\hat{}\pi'\cdot X)\sigma$. By Definition 2.3.9 we have, $\nabla \vdash ((\pi\cdot\sigma(X))\phi\sigma) \approx_\alpha ((\pi'\cdot\sigma(X))\phi'\sigma)$. Using Lemma 2.5.15 we obtain, $\nabla \vdash a\#\sigma(X)$ for each $a \in \mathtt{ds}(\nabla,(\phi\sigma)\hat{}\pi,(\phi'\sigma)\hat{}\pi')$. Using Lemma 5.3.9 we have, $\nabla \vdash Pr_i^{\approx}\sigma$ for each $Pr'_i \in \Psi(\phi\hat{}\pi,\phi\hat{}\pi,\varnothing)^A_X$ where $Pr_i^{\approx}$ is Pr'_i with constraints of form $s\ ?\approx?\ t$ replaced by constraints of form $s \approx_\alpha t$. The result follows by Definition 4.1.8.

- The case $(?\approx\bot)$. Suppose $(\{s \stackrel{?}{\approx}_? t\}, \theta) \stackrel{Xs}{\Longrightarrow}_{(?\approx\bot)} \varnothing$ where $Xs = V_{RHS}(Pr_0)$ and $s \stackrel{?}{\approx}_? t$ is a clashing equality as in Definition 5.2.5. By Definition 4.1.8 we observe that, trivially, $\mathscr{U}(s \stackrel{?}{\approx}_? t) = \varnothing$.

■

Proof of Lemma 5.3.12.

Assume $(Pr_0, \theta) \Longrightarrow_{\#}^{*} (\langle Pr_0 \rangle_{\mathtt{nf}}, \theta)$ *where* Pr_0 *is a set of freshness constraints and consider a step* $(\{Pr_{i1}, \ldots, Pr_{in}\}, \theta) \Longrightarrow_{\#} (\{Pr'_{i1}, \ldots, Pr'_{im}\}, \theta)$ *by application of some freshness rule, then* $\bigcup_{k \in 1 \ldots n} \mathscr{U}(Pr_{ik}) = \bigcup_{k' \in 1 \ldots m} \mathscr{U}(Pr'_{ik'})$.

Proof. By case analysis on the set of freshness rules for relation $\Longrightarrow_{\#}$. For the sake of simplicity, suppose $k = 1$ such that $(\{Pr_{i1}, \ldots, Pr_{in}\}, \theta) = (\{Pr_{i1}\}, \theta)$ and also $Pr_{i1} = \{C\}$, that is, Pr_{i1} contains only one problem. Since a solution to a matching problem is also a solution to a unification problem, we apply directly Definition 4.1.8 whenever required without loss of generality (recall that $\mathscr{D}om(\theta) \cap V_{RHS}(Pr_0) = \varnothing$)

- The case $(\#\bot)$. Suppose $Pr_{i1} = \{a\#a\}$. Then, $a\#a \stackrel{(\#\bot)}{\Longrightarrow} \bot$.

  Then, by the definition of unsolvable unification problem, Definition 4.1.8 and Definition 5.2.6 we observe that $\mathscr{U}(a\#a) = \varnothing$ and the result follows trivially.

- The case $(\#\mathbf{ab})$. Suppose $Pr_{i1} = \{a\#b\}$. Then, $(a\#b, \theta) \stackrel{(\#\mathbf{ab})}{\Longrightarrow} (\varnothing, \theta)$.

  Suppose also that $(\mathbf{F}, \sigma) \in \mathscr{U}(\varnothing)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash \varnothing\theta$. by Definition 2.3.12, $\varnothing\theta = \varnothing$. Using $(\#\mathbf{ab})$, $\nabla \vdash a\#b$ and by Definition 2.3.9, $b = b\theta$. The result follows by application of Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#b)$. By Definition 4.1.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#b\theta$ where $b\theta = b$ by Definition 2.3.9. Using $(\#\mathbf{ab})$, $\nabla \vdash \varnothing$. By Definition 2.3.12, $\varnothing\theta = \varnothing$. The result follows by application of Definition 4.1.8.

- The case $(\#[\mathbf{a}])$. Suppose $Pr_{i1} = \{a\#[a]s\}$. Then, $(a\#[a]s, \theta) \stackrel{(\#[\mathbf{a}])}{\Longrightarrow} (\varnothing, \theta)$.

  Suppose also that $(\mathbf{F}, \sigma) \in \mathscr{U}(\varnothing)$. By Definition 4.1.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash \varnothing\theta$. By Definition 2.3.12, $\varnothing\theta = \varnothing$. Using $(\#[\mathbf{a}])$, $\nabla \vdash a\#[a]s\theta$. By Definition 2.3.9, $\nabla \vdash a\#[a]s\theta \approx_\alpha a\#([a]s)\theta$. The result follows by application of Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#[a]s)$. By Definition 4.1.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#([a]s)\theta$. By Definition 2.3.9, $\nabla \vdash a\#([a]s)\theta = a\#[a]s\theta$. Using $(\#[\mathbf{a}]\mathbf{s})$, $\nabla \vdash \varnothing$. By Definition 2.3.12, $\varnothing\theta = \varnothing$. The result follows by application of Definition 4.1.8.

- The case (#[**b**]). Suppose $Pr_{i1} = \{a\#[b]s\}$. Then, $(a\#[b]s, \theta) \stackrel{(\#[\mathbf{b}])}{\Longrightarrow} (a\#s, \theta)$.

  Suppose also that $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#s)$. Then, by Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#s\theta$. Using (#[**b**]), $\nabla \vdash a\#[b]s\theta$. By Definition 2.3.9, $\nabla \vdash a\#[b]s\theta \approx_\alpha a\#([b]s)\theta$. The result follows by application of Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#[b]s)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#([b]s)\theta$. By Definition 2.3.9, $\nabla \vdash a\#([b]s)\theta = a\#[b]s\theta$. Using (#[**b**]), $\nabla \vdash a\#s\theta$. The result follows by application of Definition 4.1.8.

- The case (#**f**). Suppose $Pr_{i1} = \{a\#fs\}$. Then, $(a\#fs, \theta) \stackrel{(\#\mathbf{f})}{\Longrightarrow} (a\#s, \theta)$.

  Suppose also that $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#s)$. Then, by Definition 4.1.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#s\theta$. Using (#**f**), $\nabla \vdash a\#fs\theta$. By Definition 2.3.9, $\nabla \vdash a\#fs\theta \approx_\alpha a\#(fs)\theta$. The result follows by application of Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#fs)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#(fs)\theta$. By Definition 2.3.9, $\nabla \vdash a\#(fs)\theta \approx_\alpha a\#fs\theta$. Using (#**f**) we derive $\nabla \vdash a\#s\theta$. The result follows by application of Definition 4.1.8.

- The case (#**tupl**). Suppose $Pr_{i1} = \{a\#(s_1, \ldots, s_n)\}$. Then, $(a\#(s_1, \ldots, s_n), \theta) \stackrel{(\#\mathbf{tupl})}{\Longrightarrow} (a\#s_1, \ldots, a\#s_n, \theta)$.

  Suppose also that $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#s_1, \ldots, a\#s_n)$. Then, by Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#s_1\theta, \ldots, a\#s_n\theta$. Using (#**tupl**) we derive $\nabla \vdash a\#(s_1\theta, \ldots, s_n\theta)$. By Definition 2.3.9, $\nabla \vdash a\#(s_1\theta, \ldots, s_n\theta) \approx_\alpha a\#(s_1, \ldots, s_n)\theta$. The result follows by application of Definition 4.1.8.

  Otherwise, suppose $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#(s_1, \ldots, s_n))$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#(s_1, \ldots, s_n)\theta$. By Definition 2.3.9, $\nabla \vdash a\#(s_1, \ldots, s_n)\theta \approx_\alpha a\#(s_1\theta, \ldots, s_n\theta)$. Using (#**tupl**) we derive $\nabla \vdash a\#s_1\theta, \ldots, a\#s_n\theta$. The result follows by application of Definition 4.1.8.

- The case (#**X**). Suppose $Pr_{i1} = \{a\#\phi\hat{}\pi\cdot X\}$. Then, $(a\#\phi\hat{}\pi\cdot X, \theta) \stackrel{(\#\mathbf{X})}{\Longrightarrow} (\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle), \theta)$ where $\phi \neq \mathtt{Id}$ or $\pi \neq \mathtt{Id}$ by the side conditions imposed to the rule. By Definition 3.3.4 we observe that each $\langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle$ is a DNF of consistent freshness constraints, where $b \in (\mathscr{D}om(\phi) \cup \{a\})$. Then, following Lemma 3.3.8, it is the case that each $\Delta \in \mathbf{S}nf(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle))$ is a conjunctive clause of consistent freshness constraints.

Suppose also that $(\mathbf{F}_i, \theta) \in \mathscr{U}(\Delta_i)$ for each $\Delta \in \mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle))$. By Definition 4.1.8, $\forall \nabla_{ij} \in \mathbf{F}_i.\nabla_{ij} \vdash \Delta\theta$. Using Lemma 5.3.10 we have, $\nabla_{ij} \vdash n\#\theta(X)$ for each $n \in \mathtt{fresh}(\nabla, a, (\phi\theta)\hat{}\pi)$. Applying Lemma 2.5.12 on the unpacked definition of $\mathtt{fresh}(\Delta_i, a, \phi\hat{}\pi)$ (see Equation 2.2). We are now able to apply Lemma 2.5.16 to obtain $\nabla_{ij} \vdash a\#(\phi\theta)\hat{}\pi\cdot\theta(X)$. By application of Definition 2.3.9 we have, $\nabla_{ij} \vdash a\#(\phi\hat{}\pi\cdot X)\theta$ and the result follows by Definition 4.1.8.

Otherwise, suppose $(\mathbf{F}, \sigma) \in \mathscr{U}(a\#\phi\hat{}\pi\cdot X)$. By Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash a\#(\phi\hat{}\pi\cdot X)\theta$. Applying Definition 2.3.9 we have $\nabla \vdash a\#((\pi\cdot\theta(X)))\phi\theta$. By Lemma 2.5.17, $\nabla \vdash n\#\theta(X)$ for each $n \in \mathtt{fresh}(\nabla, a, (\phi\theta)\hat{}\pi)$. Using Lemma 5.3.10 we have, $\nabla \vdash \Delta_i\theta$ for each $\Delta_i \in \mathbf{S}_{nf}(\mathtt{DNF}(\bigwedge_{b \in (\mathscr{D}om(\phi) \cup \{a\})} \langle\, \pi^{-1}(b), a\#\phi(b), X \,\rangle))$. The result follows by Definition 4.1.8.

■

Proof of Lemma 5.3.14.

Let $\phi\hat{}\pi\cdot X$ and t be a pair of terms such that $V(\phi\hat{}\pi\cdot X) \cap V(t) = \varnothing$, σ an idempotent v-substitution such that $\sigma(X) \neq X$ and $\mathscr{D}om(\sigma) \cap V(t) = \varnothing$, and let ∇ be a freshness context such that σ satisfies ∇. If $\nabla \vdash (\phi\hat{}\pi\cdot X)\sigma \approx_\alpha t$, then $\nabla \vdash u([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha u\sigma$ where u is $Z \in \mathscr{D}om(\sigma) \setminus \{X\}$ or $u = \phi\hat{}\pi\cdot X$ and s is some term in $\mathtt{Cap}(t, \mathscr{D}om(\phi))$ whenever $t \neq \phi'\hat{}\pi'\cdot Y$ (resp. in $\mathtt{Cap}(t, \mathscr{D}om(\phi)) \cup \{\pi'\cdot Y\}$ whenever $t = \phi'\hat{}\pi'\cdot Y$).

Proof. For any variable $Z \in \mathscr{D}om(\sigma)$ such that $Z \neq X$, the result follows trivially. The interesting case is when applying the v-substitution to $\phi\hat{}\pi\cdot X$. Then, we need to show that

(*a*) $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha t$ for some $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$ whenever $t \neq \phi'\hat{}\pi'\cdot Y$;

(*b*) $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha \phi'\hat{}\pi'\cdot Y$ for some $s \in \mathtt{Cap}(\phi'\hat{}\pi'\cdot Y, \mathscr{D}om(\phi))$ or $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot(\pi'\cdot Y)] \bullet \sigma) \approx_\alpha \phi'\hat{}\pi'\cdot Y$.

Using Definition 2.3.9 we have,

(*a*) $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha s(\phi([X \mapsto \pi^{-1}\cdot s] \bullet \sigma))$;

(*b*) $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha s(\phi([X \mapsto \pi^{-1}\cdot s] \bullet \sigma))$ and $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot(\pi'\cdot Y)] \bullet \sigma) \approx_\alpha \phi([X \mapsto \pi^{-1}\cdot(\pi'\cdot Y)] \bullet \sigma)\hat{}\pi'\cdot Y$.

Hence, by application of the Transitive property from Theorem 2.5.8, we are left to prove that

(*a*) $\nabla \vdash s(\phi([X \mapsto \pi^{-1}\cdot s] \bullet \sigma)) \approx_\alpha t$ for some $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$;

(*b*) $\nabla \vdash s(\phi([X \mapsto \pi^{-1}\cdot s] \bullet \sigma)) \approx_\alpha \phi'\hat{}\pi'\cdot Y$ for some $s \in \mathtt{Cap}(\phi'\hat{}\pi'\cdot Y, \mathscr{D}om(\phi))$ or $\nabla \vdash \phi([X \mapsto \pi^{-1}\cdot(\pi'\cdot Y)] \bullet \sigma)\hat{}\pi'\cdot Y \approx_\alpha \phi'\hat{}\pi'\cdot Y$.

- For part (a).

  We distinguish two cases to prove, the case where $\phi = \mathtt{Id}$ and the case where $\phi \neq \mathtt{Id}$.

  • For the case where $\phi = \mathtt{Id}$.

  By Definition 5.2.3 we have, $\mathtt{Cap}(t, \varnothing) = \{t\}$ such that $s = t$. By Definition 2.3.9 we have, $\nabla \vdash t(\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma)) \approx_\alpha t$. By the Reflexive property, $\nabla \vdash t \approx_\alpha t$ always. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha t$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

  • For the case where $\phi \neq \mathtt{Id}$.

  For the sake of clarity, assume that $\phi(a) \neq a$ for atom a and $\phi(b) = b$ for $b \in (\mathscr{A} \setminus \{a\})$ (i.e., ϕ is a singleton) and also that $t|_k = a$ is the only occurrence of atom a in t, for some position $k \in \mathscr{P}os(t)$. More general cases are built by induction.

  We distinguish two cases, the case where $s = t$ and the case where $s \in \{t[a \cdots a]_{p_1 \cdots p_n} \mid a \in \mathscr{D}om(\phi), p_i \in \mathscr{P}os(t), 1 \leq i \leq n\}$. We assume without loss of generality that $i = 1$, i.e., $s = t[a]_p$ for some $t[a]_p \in (\{t[a]_p \mid a \in \mathscr{D}om(\phi), p \in \mathscr{P}os(t)\})$. Other cases where $i > 1$ are built by induction.

  1. Suppose $s = t$.

     Then, either $\nabla \vdash a\#t$ or $a \in supp(t)$ (see Definition 1.2.14), that is, a occurs unabstracted in t.

     (a) Suppose that $\nabla \vdash a\#t$.
     Then, by Definition 2.3.2 we obtain, $\nabla \vdash t(\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma)) \approx_\alpha t$ and $\nabla \vdash t \approx_\alpha t$ always. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha t$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

     (b) Otherwise, if a occurs in t unabstracted at some position k, then $|t| < |t(\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma))|$ whenever $\nabla \vdash t|_k\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma) \not\approx_\alpha t|_k$. Hence, it can only be that $\nabla \vdash t|_k\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma) \approx_\alpha t|_k$ such that $\nabla \vdash t(\phi([X \mapsto \pi^{-1}\cdot t] \bullet \sigma)) \approx_\alpha t$. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s] \bullet \sigma) \approx_\alpha t$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

  2. Otherwise, suppose $s = t[a]_p$ for some $t[a]_p \in \{t[a]_p \mid a \in \mathscr{D}om(\phi), p \in \mathscr{P}os(t)\}$. Suppose also $t|_p = v$ and $\exists p' \in \mathscr{P}os(t)$ (e.g.: $p' = \varepsilon$) such that $t|_{p'} = w$ and

$p' \leq p$ and $v \lhd w$, that is, v occurs in term w. Then, the judgement we need to prove is now represented as $\nabla \vdash (t[a]_p)(\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)) \approx_\alpha t[v]_p$.

There are two cases to show, the case where $\nabla \vdash a\#t[v]_p$ and the case where $a \in supp(t[v])_p$ (see Definition 1.2.14), that is, a occurs unabstracted in t.

(a) Suppose $\nabla \vdash a\#t[v]_p$.

Now, we have two more cases to show, either $(w = [a]l \wedge w \neq v)$ or $(w \neq [a]l \vee w = v)$.

i. Suppose that $w = [a]l \wedge w \neq v$ (hence $v \lhd l$).

Then, by Definition 2.3.2 we have, $\nabla \vdash (t[a]_p)(\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)) \approx_\alpha t[a]_p$. Hence, it can only be that $\nabla \vdash a \approx_\alpha v$ such that, by Corollary 2.5.20, $\nabla \vdash t[a]_p \approx_\alpha t[v]_p$. Then, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[a]_p$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

ii. Suppose that $(w \neq [a]l \vee w = v)$.

Then, by Definition 2.3.2 we have $\nabla \vdash t[a]_p(\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p$. Then, it can only be that, at position p, $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha v$ such that, by Corollary 2.5.20, $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha t[v]_p$. Then, by the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ where $t \equiv t[v]_p$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by another application of the Transitive property from Theorem 2.5.8.

(b) Otherwise, suppose $a \in supp(t)$ (see Definition 1.2.14), that is, a occurs unabstracted in t at position k (by the assumption).

We show two cases with respect to the structure of w, either $(w = [a]l \wedge w \neq v)$ or $(w \neq [a]l \vee w = v)$.

i. Suppose $w = [a]l \wedge w \neq v$ (hence $v \lhd l$).

By the assumption we observe there is only one occurrence of atom a in t occurring at position k and thus $t|_k \not\lhd w$ since $t|_k$ occurs unabstracted in $t\sigma$. Then, it must be the case that $k \parallel p$ so that we must prove $\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha (t[v]_p)[a]_k$. Then, by Definition 2.3.2 we obtain, $\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha ((t[a]_p)[\pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_k)$. Then, notice that $|(t|_k\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma))| = |(t|_k)|$ only for the case where

$\nabla \vdash t|_k \phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t|_k$. Further, since v is in the scope of $[a]$- by the assumption, it can only be that $\nabla \vdash a \approx_\alpha v$ at position p. Therefore, by the Transitive property from Theorem 2.5.8 it follows that $\nabla \vdash \pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha a \approx_\alpha v$ and applying Corollary 2.5.20 we have, $\nabla \vdash ((t[a]_p)[\pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_k) \approx_\alpha (t[v]_p)[a]_k$. Then, by the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ where $t \equiv t[v]_p$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash ((t[a]_p)[\pi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_k) \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by the Transitive property from Theorem 2.5.8.

ii. Otherwise, suppose that $(w \neq [a]l \vee w = v)$.
Then, either $t|_k \lhd v$ or $t|_k \not\lhd v$ where $t|_k = a$ by the assumption.
• Suppose that $t|_k \lhd v$.
Then, one has to prove $\nabla \vdash (t[a]_p)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[v]_p$. By Definition 2.3.2 we have, $\nabla \vdash (t[a]_p)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p)$. Then, it must be the case that $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma) \approx_\alpha v$ at position p such that, by Corollary 2.5.20, we obtain $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha t[v]_p$. Then, by the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ where $t \equiv t[v]_p$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)] \bullet \sigma)]_p \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$. The result follows by another application of the Transitive property from Theorem 2.5.8.
• Otherwise, suppose $t|_k \not\lhd v$.
Then, one has to prove $\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha (t[v]_p)[a]_k$. By Definition 2.3.2 we have,
$\nabla \vdash ((t[a]_p)[a]_k)\phi([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha$
$(t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_p)[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_k)$.
Then, note that, at position p, $|(\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma))| \leq |v|$ only when $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha v$ and at position k, $|(\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma))| \leq |a|$ only when $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha a$. Therefore it must be the case that, by the Transitive property from Theorem 2.5.8, $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma) \approx_\alpha v \approx_\alpha a$ such that, by Corollary 2.5.20, we obtain $\nabla \vdash (t[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_p)[\phi(a)([X \mapsto \pi^{-1}\cdot(t[a]_p)[a]_k] \bullet \sigma)]_k) \approx_\alpha (t[v]_p)[a]_k$. Then, by the assumption and the Symmetric property from Theorem 2.5.8,

$\nabla \vdash t \approx_\alpha (\phi\hat{}\pi{\cdot}X)\sigma$ where $t \equiv (t[v]_p)[a]_k$. By the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash (t[\phi(a)([X \mapsto \pi^{-1}{\cdot}(t[a]_p)[a]_k] \bullet \sigma)]_p)[\phi(a)([X \mapsto \pi^{-1}{\cdot}(t[a]_p)[a]_k] \bullet \sigma)]_k) \approx_\alpha (\phi\hat{}\pi{\cdot}X)\sigma$. The result follows by another application of the Transitive property from Theorem 2.5.8.

- For part (b). We distinguish two cases to prove, the case where $\phi = \mathtt{Id}$ and the case where $\phi \neq \mathtt{Id}$.

  1. Case $(\phi = \mathtt{Id})$.

     Then, by application of Definition 2.3.2 we obtain, $\nabla \vdash s(\phi([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma)) \approx_\alpha s$ where $s \equiv \phi'\hat{}\pi'{\cdot}Y$ since $\mathtt{Cap}(\phi'\hat{}\pi'{\cdot}Y, \varnothing) = \{\phi'\hat{}\pi'{\cdot}Y\}$, and also $\nabla \vdash \phi([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma)\hat{}\pi'{\cdot}Y \approx_\alpha \pi'{\cdot}Y$. For the former, by the Transitive property from Theorem 2.5.8 we have, $\nabla \vdash s \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$ always. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\pi{\cdot}X)\sigma$ where $t \equiv \phi'\hat{}\pi'{\cdot}Y$. The result follows for this case by another application of the Transitive property from Theorem 2.5.8. For the latter, by rule $(\approx_\alpha \mathbf{X})$ it must be the case that $\nabla \vdash n\#Y$ for each $n \in \mathtt{ds}(\nabla, \pi', \phi'\hat{}\pi')$ so that, by Lemma 2.5.14 we obtain, $\nabla \vdash \pi'{\cdot}Y \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\pi{\cdot}X)\sigma$ where $t \equiv \phi'\hat{}\pi'{\cdot}Y$. By the Transitive property from Theorem 2.5.8, $\nabla \vdash \pi'{\cdot}Y \approx_\alpha (\pi{\cdot}X)\sigma$ The result follows for this case by another application of the Transitive property from Theorem 2.5.8.

  2. Case $(\phi \neq \mathtt{Id})$.

     - We begin by proving case $\nabla \vdash \phi([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma)\hat{}\pi'{\cdot}Y \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$. By rule $(\approx_\alpha \mathbf{X})$ it must be the case that $\nabla \vdash n\#Y$ for each $n \in \mathtt{ds}(\nabla, \phi([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma)\hat{}\pi', \phi'\hat{}\pi')$ so that, by Lemma 2.5.14 we obtain, $\nabla \vdash \phi([X \mapsto \pi^{-1}{\cdot}(\pi'{\cdot}Y)] \bullet \sigma)\hat{}\pi'{\cdot}Y \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$ as expected. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\pi{\cdot}X)\sigma$ where $t \equiv \phi'\hat{}\pi'{\cdot}Y$. By the Transitive property from Theorem 2.5.8, $\nabla \vdash \pi'{\cdot}Y \approx_\alpha (\pi{\cdot}X)\sigma$ The result follows for this case by another application of the Transitive property from Theorem 2.5.8.
     - To prove case $\nabla \vdash s(\phi([X \mapsto \pi^{-1}{\cdot}s] \bullet \sigma)) \approx_\alpha \phi'\hat{}\pi'{\cdot}Y$ for some $s \in \mathtt{Cap}(\phi'\hat{}\pi'{\cdot}Y, \mathscr{D}om(\phi))$ we proceed as follows. For the sake of clarity, assume that $\phi(a) \neq a$ for atom a and $\phi(b) = b$ for $b \in (\mathscr{A} \setminus \{a\})$ (i.e., ϕ is a singleton). Similarly, assume that $\phi'(b) \neq b$ for atom b and $\phi'(a) = a$ for $a \in (\mathscr{A} \setminus \{b\})$ (i.e., ϕ' is also a singleton). More general cases are built by induction. Then, by Definition 5.2.3 and the structure of $\phi'\hat{}\pi'{\cdot}Y$,

$\mathtt{Cap}(\phi'^\wedge\pi'\cdot Y, \mathscr{D}om(\phi)) = \{a, [b \mapsto a]^\wedge\pi'\cdot Y, \phi'^\wedge\pi'\cdot Y\}$. No other cases are possible. We look at each case separately.

* $(s = a)$.
  By Definition 2.3.2, $\nabla \vdash a(\phi([X \mapsto \pi^{-1}\cdot a] \bullet \sigma)) \approx_\alpha \phi(a)([X \mapsto \pi^{-1}\cdot a] \bullet \sigma)$. Then, it can only be that $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot a] \bullet \sigma) \approx_\alpha \phi'^\wedge\pi'\cdot Y$. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi^\wedge\pi\cdot X)\sigma$ where $t \equiv \phi'^\wedge\pi'\cdot Y$. By the Transitive property from Theorem 2.5.8, $\nabla \vdash \phi(a)([X \mapsto \pi^{-1}\cdot a] \bullet \sigma) \approx_\alpha (\phi^\wedge\pi\cdot X)\sigma$. The result follows for this case by another application of the Transitive property from Theorem 2.5.8.
* $(s = [b \mapsto a]^\wedge\pi'\cdot Y)$.
  By application of Definition 2.3.2,
  $\nabla \vdash ([b \mapsto a]^\wedge\pi'\cdot Y)\phi([X \mapsto \pi^{-1}\cdot([b \mapsto a]^\wedge\pi'\cdot Y)] \bullet \sigma) \approx_\alpha$
  $([b \mapsto \phi(a)([X \mapsto \pi^{-1}\cdot([b \mapsto a]^\wedge\pi'\cdot Y)] \bullet \sigma); a \mapsto \phi(a)([X \mapsto \pi^{-1}\cdot([b \mapsto a]^\wedge\pi'\cdot Y)] \bullet \sigma)])^\wedge\pi'\cdot Y$. Fix $\phi'' = ([b \mapsto \phi(a)([X \mapsto \pi^{-1}\cdot([b \mapsto a]^\wedge\pi'\cdot Y)] \bullet \sigma); a \mapsto \phi(a)([X \mapsto \pi^{-1}\cdot([b \mapsto a]^\wedge\pi'\cdot Y)] \bullet \sigma)])$ for short. Now, following $(\approx_\alpha \mathbf{X})$, it must be the case that $\mathtt{ds}(\nabla, \phi''^\wedge\pi', \phi'^\wedge\pi')\#Y$ so that by Lemma 2.5.14 we obtain, $\nabla \vdash \phi''^\wedge\pi'\cdot Y \approx_\alpha \phi'^\wedge\pi'\cdot Y$ as expected. By the assumption and the Symmetric property from Theorem 2.5.8, $\nabla \vdash t \approx_\alpha (\phi^\wedge\pi\cdot X)\sigma$ where $t \equiv \phi'^\wedge\pi'\cdot Y$. By the Transitive property from Theorem 2.5.8, $\nabla \vdash \phi''^\wedge\pi'\cdot Y \approx_\alpha (\phi^\wedge\pi\cdot X)\sigma$. The result follows for this case by another application of the Transitive property from Theorem 2.5.8.
* $(s = \phi'^\wedge\pi'\cdot Y)$.
  By application of Definition 2.3.2, $\nabla \vdash (\phi'^\wedge\pi'\cdot Y)\phi([X \mapsto \pi^{-1}\cdot([b \mapsto a]^\wedge\pi'\cdot Y)] \bullet \sigma) \approx_\alpha (\phi' \bullet \phi([X \mapsto \pi^{-1}\cdot([b \mapsto a]^\wedge\pi'\cdot Y)] \bullet \sigma))^\wedge\pi'\cdot Y$. The rest of the proof follows similarly to the previous case above and thus omitted.

Hence the property holds. ■

Proof of Lemma 5.3.15.

Assume $(Pr_0, \mathtt{Id}) \overset{Xs}{\Longrightarrow}_* \langle Pr\rangle_{\mathtt{nf}_{?\approx?}}$ *where* $Xs = V_{RHS}(Pr_0)$*, and consider a step* $(Pr, \theta) \overset{Xs}{\Longrightarrow} \bigcup_{i\in I}(Pr'_i, \theta \bullet \theta'_i)$ *via some instantiating rule,*

1. *If* $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr)$*, then* $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr'_i)$ *and* $\forall \nabla \in \mathbf{F}.\nabla \vdash v(\theta'_i \bullet \sigma) \approx_\alpha v\sigma$ *for some* $i \in I$ *and every constraint* $v\ {}_?\!\approx_?\ w$ *such that* $v\ {}_?\!\approx_?\ w \in Pr$.

2. *For all* $i \in I$*, if* $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr'_i)$*, then* $(\mathbf{F}, \theta'_i \bullet \sigma) \in \mathscr{U}(Pr)$.

Proof. There are two instantiating rules to examine, rule ($_?\approx$**Inst**) and rule ($_?\approx$**XY**).

For rule ($_?\approx$**Inst**) there is the following instantiation step:
Let $Pr = \{\phi\hat{}\pi\cdot X \ {}_?\approx_?\ t\} \cup Pr''$ where $t \neq \phi'\hat{}\pi'\cdot Y$ for some $Y \in \mathscr{X}$ and θ a v-substitution. Then, $(Pr,\theta) \overset{X\notin Xs}{\Longrightarrow}_{(_?\approx\textbf{Inst})} \bigcup_{s\in \mathtt{Cap}(t,\mathscr{D}om(\phi))} (\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \ {}_?\approx_?\ t\} \cup Pr''[X \mapsto \pi^{-1}\cdot s])$ where $X \notin V(s)$ by definition of matching solution as in Definition 5.1.1 and definition of function Cap as in Definition 5.2.3 for each $s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$.

For rule ($_?\approx$**XY**), there is the following instantiation step:
Let $Pr = \{\phi\hat{}\pi\cdot X \ {}_?\approx_?\ \phi'\hat{}\pi'\cdot Y\} \cup Pr''$ for some $X,Y \in \mathscr{X}$ such that $X \neq Y$ and θ a v-substitution. Then, $(Pr,\theta) \overset{X\notin Xs}{\Longrightarrow}_{(_?\approx\textbf{XY})} \bigcup_{s\in(\mathtt{Cap}(\phi'\hat{}\pi'\cdot Y,\mathscr{D}om(\phi))\cup\{\pi'\cdot Y\})} (\{s(\phi[X \mapsto \pi^{-1}\cdot s]) \ {}_?\approx_?\ \phi'\hat{}\pi'\cdot Y\} \cup Pr''[X \mapsto \pi^{-1}\cdot s], \theta \bullet [X \mapsto \pi^{-1}\cdot s])$ where $X \notin V(s)$ by definition of a matching solution as in Definition 5.1.1 and definition of function Cap as in Definition 5.2.3 for each $s \in s \in \mathtt{Cap}(\phi'\hat{}\pi'\cdot Y,\mathscr{D}om(\phi))$.

1. For the first part, we look separately at each of the two instantiating rules.

($_?\approx$**Inst**)

Suppose that $(\mathbf{F},\sigma) \in \mathscr{U}(\phi\hat{}\pi\cdot X \ {}_?\approx_?\ t, Pr'')$. By Definition 4.1.8 we have, $\forall\nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi\cdot X)\sigma \approx_\alpha t\sigma, Pr''\sigma$ where, by Definition 5.1.1, it is the case that $V(\phi\hat{}\pi\cdot X)\cap V(t) = \varnothing$ and $\mathscr{D}om(\sigma)\cap V(t) = \varnothing$ so that, by Lemma 5.3.14 we obtain, $\forall\nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ and $\forall\nabla \in \mathbf{F}.\nabla \vdash ([X \mapsto \pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha \sigma$ for all $Y \in \mathscr{D}om(\sigma)\setminus\{X\}$ and some term s such that $s \in \mathtt{Cap}(t,\mathscr{D}om(\phi))$. By composition of v-substitutions we have, $\forall\nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha ((\phi\hat{}\pi\cdot X)[X \mapsto \pi^{-1}\cdot s])\sigma$. By Definition 2.3.9 we have,
$\forall\nabla \in \mathbf{F}.\nabla \vdash ((\phi\hat{}\pi\cdot X)[X \mapsto \pi^{-1}\cdot s])\sigma \approx_\alpha (s(\phi[X \mapsto \pi^{-1}\cdot s]))\sigma$. As a result of $\mathscr{D}om(\sigma) \subseteq V(Pr)$ and applying Lemma 2.5.13 we obtain, $\forall\nabla \in \mathbf{F}.\nabla \vdash Pr''\sigma \approx_\alpha Pr''([X \mapsto \pi^{-1}\cdot s]\bullet\sigma)$. By composition of v-substitutions we have, $\forall\nabla \in \mathbf{F}.\nabla \vdash Pr''([X \mapsto \pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha (Pr''([X \mapsto \pi^{-1}\cdot s])\sigma$. Using the Transitive property from Theorem 2.5.8 we have, $\forall\nabla \in \mathbf{F}.\nabla \vdash (s(\phi[X \mapsto \pi^{-1}\cdot s]))\sigma \approx_\alpha t\sigma, (Pr''([X \mapsto \pi^{-1}\cdot s])\sigma)$. The result follows by Definition 4.1.8.

($_?\approx$**XY**)

Suppose that $(\mathbf{F},\sigma) \in \mathscr{U}(\phi\hat{}\pi\cdot X \approx_\alpha \phi'\hat{}\pi'\cdot Y, Pr'')$. By Definition 4.1.8 we have, $\forall\nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi\cdot X)\sigma \approx_\alpha (\phi'\hat{}\pi'\cdot Y)\sigma, Pr''\sigma$ where, by Definition 5.1.1, it is the case that $V(\phi\hat{}\pi\cdot X)\cap V(\phi'\hat{}\pi'\cdot Y) = \varnothing$ and $\mathscr{D}om(\sigma)\cap V(\phi'\hat{}\pi'\cdot Y) = \varnothing$ so that, by Lemma 5.3.14 we obtain, $\forall\nabla \in \mathbf{F}.\nabla \vdash (\phi\hat{}\pi\cdot X)([X \mapsto \pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha (\phi\hat{}\pi\cdot X)\sigma$ and $\forall\nabla \in \mathbf{F}.\nabla \vdash ([X \mapsto \pi^{-1}\cdot s]\bullet\sigma) \approx_\alpha \sigma$ for all $Z \in \mathscr{D}om(\sigma)\setminus\{X\}$ and some

term $s \in (\mathtt{Cap}(t, \mathscr{D}om(\phi)) \cup \{\pi' \cdot Y\})$. The rest follows as in the previous case and thus omitted.

2. For the second part, we also look separately at each of the two instantiating rules ($_?\approx$**Inst**) and ($_?\approx$**XY**).

($_?\approx$**Inst**) Suppose that, for any $s \in \mathtt{Cap}(t, \mathscr{D}om(\phi))$, $(\mathbf{F}, \sigma) \in \mathscr{U}(Pr'_s)$ where $Pr'_s = s(\phi\theta') \;_?\approx_?\; t, Pr''\theta'$ and $\theta' = [X \mapsto \pi^{-1} \cdot s]$. By application of Definition 4.1.8 we have $\forall \nabla \in \mathbf{F}.\nabla \vdash (s(\phi\theta'))\sigma \approx_\alpha t\sigma, (Pr''\theta')\sigma$ where by Definition 5.1.1, $\mathscr{D}om(\sigma) \cap V(t) = \varnothing$. Thus by Lemma 2.5.13 we have, trivially, $\forall \nabla \in \mathbf{F}.\nabla \vdash t\sigma \approx_\alpha t(\theta' \bullet \sigma)$. Applying composition of v-substitutions, $\forall \nabla \in \mathbf{F}.\nabla \vdash t(\theta' \bullet \sigma) \approx_\alpha (t\theta')\sigma$ and by the Transitive property from Theorem 2.5.8 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash (s(\phi\theta'))\sigma \approx_\alpha (t\theta')\sigma, (Pr''\theta')\sigma$. Now, by claim 2. of Lemma 2.5.4 we have, $\forall \nabla \in \mathbf{F}.\nabla \vdash (s(\phi\theta'))\sigma \approx_\alpha (\pi \circ \pi^{-1}) \cdot ((s(\phi\theta'))\sigma)$. Applying Property 2.5.10 we obtain, $\forall \nabla \in \mathbf{F}.\nabla \vdash (\pi \circ \pi^{-1}) \cdot ((s(\phi\theta'))\sigma) \approx_\alpha ((\pi \circ \pi^{-1}) \cdot (s(\phi\theta')))\sigma$. Applying Property 2.5.9 we have,
$\forall \nabla \in \mathbf{F}.\nabla \vdash ((\pi \circ \pi^{-1}) \cdot (s(\phi\theta')))\sigma \approx_\alpha (((\pi \circ \pi^{-1}) \cdot s)((\pi \circ \pi^{-1}) \cdot (\phi\theta')))\sigma$.
Since $(\pi \circ \pi^{-1}) = \mathtt{Id}$ we observe that,
$\forall \nabla \in \mathbf{F}.\mathtt{ds}(\nabla, ((\pi \circ \pi^{-1}) \cdot (\phi\theta'))\hat{}(\pi \circ \pi^{-1}), (\phi\theta')\hat{}(\pi \circ \pi^{-1})) = \varnothing$ such that, by Lemma 2.5.14 we obtain,
$\forall \nabla \in \mathbf{F}.\nabla \vdash (((\pi \circ \pi^{-1}) \cdot s)((\pi \circ \pi^{-1}) \cdot (\phi\theta')))\sigma \approx_\alpha (((\pi \circ \pi^{-1}) \cdot s)(\phi\theta'))\sigma$. By Definition 2.1.8, $(((\pi \circ \pi^{-1}) \cdot s)(\phi\theta'))\sigma = ((\pi \cdot (\pi^{-1} \cdot s))(\phi\theta'))\sigma$. By the assumption, $((\pi \cdot (\pi^{-1} \cdot s))(\phi\theta'))\sigma = ((\pi \cdot \theta'(X))(\phi\theta'))\sigma$. Using Definition 2.3.9, $\forall \nabla \in \mathbf{F}.\nabla \vdash ((\pi \cdot \theta'(X))(\phi\theta'))\sigma \approx_\alpha ((\phi\hat{}\pi \cdot X)\theta')\sigma$. Applying the Transitive property from Theorem 2.5.8 we obtain, $\forall \nabla \in \mathbf{F}.\nabla \vdash ((\phi\hat{}\pi \cdot X)\theta')\sigma \approx_\alpha (t\theta')\sigma, (Pr''\theta')\sigma$. By Definition 4.1.8 we obtain, $(\mathbf{F}', \sigma) \in \mathscr{U}((\phi\hat{}\pi \cdot X)\theta' \approx_\alpha t\theta', Pr''\theta')$. The result follows by Lemma 5.3.13.

($_?\approx$**XY**) Suppose that, $(\mathbf{F}_s, \sigma_s) \in \mathscr{U}(s(\phi[X \mapsto \pi^{-1} \cdot s]) \;_?\approx_?\; \phi'\hat{}\pi' \cdot Y, Pr''[X \mapsto \pi^{-1} \cdot s])$ for any $s \in (\mathtt{Cap}(\phi'\hat{}\pi' \cdot Y, \mathscr{D}om(\phi)) \cup \{\pi' \cdot Y\})$. By application of Definition 4.1.8 we have, $\forall \nabla_s \in \mathbf{F}_s.\nabla_s \vdash (s(\phi[X \mapsto \pi^{-1} \cdot s]))\sigma_s \approx_\alpha (\phi'\hat{}\pi' \cdot Y)\sigma_s, (Pr''[X \mapsto \pi^{-1} \cdot s])\sigma_s$ where by Definition 5.1.1, $\mathscr{D}om(\sigma_s) \cap V(\phi'\hat{}\pi' \cdot Y) = \varnothing$. The result now follows similarly to previous case ($_?\approx$**Inst**).

■

Proof of Lemma 5.3.16.

Suppose $\Delta \vdash X\sigma \approx_\alpha s\sigma$ and $X \notin V(s)$ for some variable X, v-substitution σ, freshness context Δ and term s. Then,

- $\Delta \vdash X\sigma \approx_\alpha X([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$, *and also*
- $\Delta \vdash Y\sigma \approx_\alpha Y([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$ *for* $Y \in \mathscr{X} \setminus \{X\}$.

Proof. For the first claim, by Definition 2.3.9 we have, $\Delta \vdash X([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}}) \approx_\alpha s\sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}}$. Using Lemma 2.5.13 and assumption $X \notin V(s)$ we obtain, $\Delta \vdash s\sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}} \approx_\alpha s\sigma$. By the Transitive property from Theorem 2.5.8 we have, $\Delta \vdash X([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}}) \approx_\alpha s\sigma$. Applying the Symmetric property from the same Theorem 2.5.8, $\Delta \vdash s\sigma \approx_\alpha X([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$. The result follows by assumption $\Delta \vdash X\sigma \approx_\alpha s\sigma$ and one more application of the Transitive property from Theorem 2.5.8.

For the second claim we have, by Definition 2.3.9, $\Delta \vdash Y([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}}) \approx_\alpha Y\sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}}$. Using Lemma 2.5.13 and the fact that $X \neq Y$ we obtain, $\Delta \vdash Y\sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}} \approx_\alpha Y\sigma$. By the Transitive property from Theorem 2.5.8 we have, $\Delta \vdash Y([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}}) \approx_\alpha Y\sigma$. Applying the Symmetric property from Theorem 2.5.8 we have, $\Delta \vdash Y\sigma \approx_\alpha Y([X \mapsto s] \bullet \sigma|_{\mathscr{D}om(\sigma)\setminus\{X\}})$. The result follows. ■

Appendix C

Additional Proofs of Part IV

Proof of Theorem 6.2.10.

Let p be a position, π a permutation and θ a v-substitution. Suppose that $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ using rule $R = (\nabla \vdash l \to r)$. Then,

1. *For any pair of nominal terms u, v, if $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ and $\Delta \vdash u[t]_{p'} \approx_\alpha v$ then $\Delta \vdash u[s]_{p'} \to_{\langle R,p,\pi,\theta\rangle} v$ (closure under context application);*

2. *Let $\Gamma \vdash \Delta\sigma$ where Γ is consistent. Then, it is the case that $\Gamma \vdash s\sigma \to_{\langle R,p,\pi,\theta\rangle} t\sigma$ (closure under substitution);*

3. *For any permutation π', if $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ then $\Delta \vdash \pi' \cdot s \to_{\langle R,p,(\pi' \circ \pi),(\pi' \cdot \theta)\rangle} \pi' \cdot t$ (closure under permutation).*

Proof. We consider each case separately.

- For the first claim,
  Suppose $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$. By Definition 6.2.4, it is the case that $\Delta \vdash \nabla^\pi\theta$ and $\Delta \vdash s|_p \approx_\alpha l^\pi\theta$ and $\Delta \vdash s[r^\pi\theta]_p \approx_\alpha t$. Then, by Lemma 6.2.9 we have $\Delta \vdash s \approx_\alpha s[l^\pi\theta]_p$. By the congruence property from Corollary 2.5.20 we obtain $\Delta \vdash u[s]_{p'} \approx_\alpha u[s[l^\pi\theta]_p]_{p'}$ and also $\Delta \vdash u[s[r^\pi\theta]_p]_{p'} \approx_\alpha u[t]'_p$. By application of the transitive property from Theorem 2.5.8 we obtain $\Delta \vdash u|_{p'} \approx_\alpha s \approx_\alpha s[l^\pi\theta]_p$ and also $\Delta \vdash u[s[r^\pi\theta]_p]_{p'} \approx_\alpha u[t]'_p \approx_\alpha v$.

  Hence, we have $\Delta \vdash \nabla^\pi\theta, \Delta \vdash u|_{p'} \approx_\alpha s[l^\pi\theta]_p$ and $\Delta \vdash u[s[r^\pi\theta]_p]_{p'} \approx_\alpha v$. The result follows by Definition 6.2.4.

- For the second claim,
  By Corollary 3.2.14, $\Delta\sigma$ is also consistent.

Suppose $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$. By Definition 6.2.4, it is the case that $\Delta \vdash \nabla^{\pi}\theta$ and $\Delta \vdash s|_p \approx_\alpha l^{\pi}\theta$ and $\Delta \vdash s[r^{\pi}\theta]_p \approx_\alpha t$. Following Lemma 2.5.12 we obtain $\Psi \vdash \nabla^{\pi}\theta\sigma$ and $\Psi \vdash l^{\pi}\theta\sigma \approx_\alpha s|_p\sigma$ and $\Psi \vdash (s[r^{\pi}\theta]_p)\sigma \approx_\alpha t\sigma$ for some $\Psi \in \langle\Delta\sigma\rangle_{\mathtt{nf}}$. The result follows by the Cut rule from Corollary 3.2.17 and Definition 6.2.4.

- For the third claim,
  Suppose $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$. By Definition 6.2.4, it is the case that $\Delta \vdash \nabla^{\pi}\theta$ and $\Delta \vdash s|_p \approx_\alpha l^{\pi}\theta$ and $\Delta \vdash s[r^{\pi}\theta]_p \approx_\alpha t$. By application of Lemma 2.5.5, $\Delta \vdash \pi'\cdot(\nabla^{\pi}\theta)$ and $\Delta \vdash (\pi'\cdot s)|_p \approx_\alpha \pi'\cdot(l^{\pi})\theta$ and $\Delta \vdash s'[\pi'\cdot(r^{\pi})\theta]_p \approx_\alpha \pi'\cdot t$ where s' is the term obtained by application of π' to term s everywhere except below position p. Following Property 2.5.10 and the Transitive property from Theorem 2.5.8 we have, $\Delta \vdash (\pi'\cdot\nabla^{\pi})\theta$ and $\Delta \vdash (\pi'\cdot s)|_p \approx_\alpha (\pi'\cdot l^{\pi})\theta$ and $\Delta \vdash s'[(\pi'\cdot r^{\pi})\theta]_p \approx_\alpha \pi'\cdot t$. By Lemma 6.1.6 we obtain, $\Delta \vdash (\nabla^{\pi^{\pi'}}\sigma)\theta$ and $\Delta \vdash (\pi'\cdot s)|_p \approx_\alpha (l^{\pi^{\pi'}}\sigma)\theta$ and $\Delta \vdash s'[(r^{\pi^{\pi'}}\sigma)\theta]_p \approx_\alpha \pi'\cdot t$ where $\sigma(X) = \pi'\cdot X$ for all $X \in V(l)$ (then, $X \in V(\nabla, l)$ too because of Definition 6.1.1). By composition of v-substitutions we have, $\Delta \vdash \nabla^{\pi^{\pi'}}(\sigma \bullet \theta)$ and $\Delta \vdash (\pi'\cdot s)|_p \approx_\alpha l^{\pi^{\pi'}}(\sigma \bullet \theta)$ and $\Delta \vdash s'[(r^{\pi^{\pi'}}(\sigma \bullet \theta)]_p \approx_\alpha \pi'\cdot t$. Using Lemma 6.1.5 we obtain, $\Delta \vdash \nabla^{(\pi'\circ\pi)}(\sigma \bullet \theta)$ and $\Delta \vdash (\pi'\cdot s)|_p \approx_\alpha l^{(\pi'\circ\pi)}(\sigma \bullet \theta)$ and $\Delta \vdash s'[r^{(\pi'\circ\pi)}(\sigma \bullet \theta)]_p \approx_\alpha \pi'\cdot t$. Now, by Definition 2.3.10 we have, $\Delta \vdash \nabla^{(\pi'\circ\pi)}(\pi'\cdot\theta)$ and $\Delta \vdash (\pi'\cdot s)|_p \approx_\alpha l^{(\pi'\circ\pi)}(\pi'\cdot\theta)$ and $\Delta \vdash s'[r^{(\pi'\circ\pi)}(\pi'\cdot\theta)]_p \approx_\alpha \pi'\cdot t$ where we assume without loss of generality $\mathscr{D}om(\sigma) \subseteq \mathscr{D}om(\theta)$. The result follows by Definition 6.2.4.

■

Proof of Lemma 6.2.11.
If $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$ *then* $s \to_{\langle R^{\pi},p,\mathtt{Id},\theta\rangle} t$.

Proof. Suppose $\Delta \vdash s \to_{\langle R,p,\pi,\theta\rangle} t$. By claim 3 of Theorem 6.2.10 we obtain $\Delta \vdash \pi^{-1}\cdot s \to_{\langle R,p,(\pi^{-1}\circ\pi),(\pi^{-1}\cdot\theta)\rangle} \pi^{-1}\cdot t$. By Definition 6.2.4, it is the case that $\Delta \vdash \nabla^{(\pi^{-1}\circ\pi)}(\pi^{-1}\cdot\theta)$ and $\Delta \vdash (\pi^{-1}\cdot s)|_p \approx_\alpha l^{(\pi^{-1}\circ\pi)}(\pi^{-1}\cdot\theta)$ and $\Delta \vdash s'[r^{(\pi^{-1}\circ\pi)}(\pi^{-1}\cdot\theta)]_p \approx_\alpha \pi^{-1}\cdot t$ where s' is the term obtained by application of π^{-1} to term s everywhere except below position p. By application of Lemma 6.1.5, $\Delta \vdash \nabla^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta)$ and $\Delta \vdash (\pi^{-1}\cdot s)|_p \approx_\alpha l^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta)$ and $\Delta \vdash s'[r^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta)]_p \approx_\alpha \pi^{-1}\cdot t$. Now, using Lemma 2.5.5 we obtain, $\Delta \vdash \pi\cdot(\nabla^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))$ and $\Delta \vdash \pi\cdot(\pi^{-1}\cdot s)|_p \approx_\alpha \pi\cdot(l^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))$ and $\Delta \vdash s''[\pi\cdot(r^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))]_p \approx_\alpha \pi\cdot(\pi^{-1}\cdot t)$ where s'' is the term obtained by application of π to term s' everywhere except below position p. By composition of permutations from Definition 2.1.2 we have, $\Delta \vdash \pi\cdot(\nabla^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))$ and $\Delta \vdash (\pi\circ\pi^{-1}\cdot s)|_p \approx_\alpha \pi\cdot(l^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))$ and $\Delta \vdash s''[\pi\cdot(r^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))]_p \approx_\alpha \pi\circ\pi^{-1}\cdot t$. Now, by the property of permutations it is a fact that $(\pi\circ\pi^{-1}) = \mathtt{Id}$, therefore $\Delta \vdash s|_p \approx_\alpha \pi\cdot(l^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))$ and also $\Delta \vdash s[\pi\cdot(r^{\pi^{\pi^{-1}}}(\pi^{-1}\cdot\theta))]_p \approx_\alpha t$. Following Property 2.5.10, $\Delta \vdash$

$(\pi\cdot\nabla^{\pi^{\text{-}1}})(\pi^{\text{-}1}\cdot\theta)$ and $\Delta\vdash s|_p \approx_\alpha (\pi\cdot l^{\pi^{\text{-}1}})(\pi^{\text{-}1}\cdot\theta)$ and also $\Delta\vdash s[(\pi\cdot r^{\pi^{\text{-}1}})(\pi^{\text{-}1}\cdot\theta)]_p \approx_\alpha t$. By application of Lemma 6.1.6 and the Transitive property from Theorem 2.5.8 we have, $\Delta\vdash (\nabla^{\pi^{(\pi\circ\pi^{\text{-}1})}}\sigma)(\pi^{\text{-}1}\cdot\theta)$ and $\Delta\vdash s|_p \approx_\alpha (l^{\pi^{(\pi\circ\pi^{\text{-}1})}}\sigma)(\pi^{\text{-}1}\cdot\theta)$ and also $\Delta\vdash s[(r^{\pi^{(\pi\circ\pi^{\text{-}1})}}\sigma)(\pi^{\text{-}1}\cdot\theta)]_p \approx_\alpha t$ where $\sigma(X)=\pi'\cdot X$ for all $X\in V(l)$ (then, $X\in V(\nabla,l)$ too because of Definition 6.1.1). By composition of v-substitution we obtain, $\Delta\vdash \nabla^{\pi^{(\pi\circ\pi^{\text{-}1})}}(\sigma\bullet(\pi^{\text{-}1}\cdot\theta))$ and $\Delta\vdash s|_p \approx_\alpha l^{\pi^{(\pi\circ\pi^{\text{-}1})}}(\sigma\bullet(\pi^{\text{-}1}\cdot\theta))$ and also $\Delta\vdash s[r^{\pi^{(\pi\circ\pi^{\text{-}1})}}(\sigma\bullet(\pi^{\text{-}1}\cdot\theta))]_p \approx_\alpha t$ and by Definition 2.3.10 we have, $\Delta\vdash \nabla^{\pi^{(\pi\circ\pi^{\text{-}1})}}((\pi\circ\pi^{\text{-}1})\cdot\theta)$ and $\Delta\vdash s|_p \approx_\alpha l^{\pi^{(\pi\circ\pi^{\text{-}1})}}((\pi\circ\pi^{\text{-}1})\cdot\theta)$ and also $\Delta\vdash s[r^{\pi^{(\pi\circ\pi^{\text{-}1})}}((\pi\circ\pi^{\text{-}1})\cdot\theta)]_p \approx_\alpha t$ where we assume without loss of generality that $\mathscr{D}om(\sigma)\subseteq\mathscr{D}om((\pi\cdot\theta)))$. The result follows by Definition 6.2.4 and the fact that $(\pi\circ\pi^{\text{-}1})=\mathtt{Id}$. ■